INPUT

6

L. 2800

...O PRATICO DI PROGRAMMAZIONE
...RARE E DIVERTIRSI COL COMPUTER

CORSO PRATICO DI PROGRAMMAZIONE
PER LAVORARE E DIVERTIRSI COL COMPUTER

Pubblicazione a fascicoli settimanali
edita dall'Istituto Geografico De Agostini

volume I - fascicolo 6

Direttori: Achille Boroli - Adolfo Boroli

Direzione editoriale: Mario Nilo; *settore fascicoli:* Jason Vella

Redazione dell'edizione italiana a cura della:
Logical Studio Communication
Traduzione dall'inglese a cura di: Daniel Quinn

Coordinamento grafico: Otello Geddo
Coordinamento fotografico a cura del Centro Iconografico dell'Istituto Geografico De Agostini
Direzione: Novara (28100), via Giovanni da Verrazano 15 - tel. (0321) 471201-5
Redazione: Milano (20149), via Mosè Bianchi 6 - tel. (02) 4694451

Programma di abbonamento. Condizioni di abbonamento all'intera opera in 52 fascicoli, completa di copertine e di risguardi per la confezione dei 6 volumi dell'opera:
a) in un unico versamento anticipato di L. 180 000 in Italia, L. 225 000 all'estero;
b) in 4 versamenti trimestrali consecutivi e anticipati di L. 45 250 ciascuno.
La forma di abbonamento *b* è ammessa soltanto in Italia.
Agli abbonati all'intera opera sono riservato in dono "2 cassette di videogiochi" oppure, in alternativa, "5 cassette da registrare" (Aut. Min. conc.).
I versamenti possono essere effettuati a mezzo assegno bancario oppure sul c/c postale n. 111286 intestato all'Istituto Geografico De Agostini - Novara.
Amministrazione, abbonamenti e servizio arretrati: Istituto Geografico De Agostini - Novara (28100), via Giovanni da Verrazano 15 - tel. (0321) 471201-5.
Copertine e risguardi per i volumi dell'opera saranno messi in vendita a L. 6000 la copia (L. 7500 all'estero).
Le copie arretrate saranno disponibili per un anno dal completamento dell'opera e potranno essere prenotate nelle edicole o direttamente presso l'Editore. Per i fascicoli arretrati, trascorse 12 settimane dalla loro pubblicazione, è applicato un sovrapprezzo di L. 400 sul prezzo di copertina in vigore al momento dell'evasione dell'ordine. Spedizione contro rimessa di pagamento anticipato; non vengono effettuate spedizioni contrassegno.
L'Editore si riserva la facoltà di modificare il prezzo nel corso della pubblicazione, se costretto da mutate condizioni di mercato.

Registrato presso il Tribunale di Novara n. 11 in data 19-5-1984.
Direttore responsabile: Emilio Bucciotti

Spedizione in abbonamento postale Gruppo II/70 (Autorizzazione della Direzione provinciale delle PP.TT. di Novara).
Distribuzione A. & G. Marco - Milano, via Fortezza 27 - tel. (02) 2526.
Pubblicazione a fascicoli settimanali. Esce il martedì.
Stampato in Italia - I.G.D.A. Officine Grafiche, Novara - 278411.

Referenze dei disegni e delle fotografie:
Copertina: Dave King. Pagg. 162, 164, 166 Chris Lyons. Pag. 168 Tudor Art Studio/Bernard Robinson. Pag. 170 Bernard Fallon. Pag. 172 Howard Kingsnorth. Pagg. 174, 176 Nick Farmer. Pag. 178 Nick Fermer, Howard Kingsnorth. Pag. 180 Dave King. Pag. 182 David Lloyd, Dick Ward. Pag. 184 Hussein Hussein, Martin Cleaver. Pag. 186 Hussein Hussein, Peter Dazely. Pag. 188 David Lloyd. Pag. 190 Hussein Hussein, The Picture Library.

INPUT È STUDIATA APPOSITAMENTE PER:

Lo SPECTRUM della Sinclair (versioni 16K e 48K), il COMMODORE 64, l'ELECTRON ed il BBC della ACORN, il DRAGON 32.

Comunque, molti dei programmi e dei testi sono adatti anche per: lo ZX81 della SINCLAIR, il COMMODORE VIC 20 ed il TANDY COLOUR COMPUTER con 32K ed il BASIC esteso.

I seguenti simboli identificano i programmi o le spiegazioni adatte a ciascun computer:

 SPECTRUM

 COMMODORE 64

 ELECTRON e BBC

 DRAGON 32

 ZX81 VIC 20

 TANDY TRS80 COLOUR COMPUTER

FACCIAMO UN BIG BANG

- LAMPI SULLO SCHERMO
- UN SEMPLICE PROGRAMMA PER UN BOMBARDAMENTO AEREO
- COME AGGIUNGERE FIAMME ED ESPLOSIONI

Le esplosioni fanno parte del repertorio di base del programmatore di giochi, dai combattimenti aerei alle guerre stellari. Ecco alcuni effetti grafici adatti a una vasta gamma di giochi

È facile rendere i propri giochi molto più spettacolari mediante routine che creino effetti speciali di grafica: la differenza sarà vistosa anche senza ricorrere a complessi programmi.

Come vedremo, esistono molti modi diversi per produrre strabilianti effetti visivi e tutto sta nel ricordarsi quale sia l'effetto giusto per un particolare gioco.

Qui viene proposto un effetto "fuoco" adatto, più che ai giochi spaziali, a tutti quei giochi nei quali è previsto l'incendio di edifici, auto, navi ecc.

Inoltre, poiché fiamme ed esplosioni vengono disegnate con l'uso di UDG, la loro grandezza massima è limitata e non possono essere aggiunti a un gioco qualsiasi senza prima apportare modifiche alla routine.

D'altra parte, alcuni apparecchi possiedono routine per generare lampi sullo schermo molto più adatte a giochi spaziali, con in più il vantaggio che non devono essere adattate di volta in volta.

S

Il segreto per visualizzare un'"esplosione" sullo Spectrum è tutto in una POKE che provoca fiamme e detriti. Quando l'edificio crolla, essi svaniscono gradualmente, come sarà spiegato più sotto. Prima ci servono un aereo e una bomba, perciò si trascriva questo programma:

```
10 FOR n=USR "p" to USR "q"+7
20 READ a
30 POKE n,a
40 NEXT n
50 DATA 32,16,136,154,155,8,16,32
60 DATA 0,16,16,120,28,28,0,0
```

Per creare le immagini dell'aereo e della bomba si ricorre all'UDG descritto alle pagine 38-45. Si esegua ora un RUN e, per controllare l'esattezza

delle DATA trascritte, si digiti quanto segue (senza numero di linea):

```
PRINT AT 10,15; CHR$159; "□"; CHR$160
```

che visualizzerà i due UDG. Serve poi un edificio da bombardare, ma non un nuovo programma. Dopo avere digitato quello esistente, lo si corregga così:

```
10 FOR n=USR "r" to USR "r"+7
20 READ a
30 POKE n,a
40 NEXT n
50 DATA 255,153,255,153,255,153,255,255
```

Si esegua un altro RUN e, per provare questo pezzo, si scriva (senza numero di linea):

```
PRINT AT 20, 15; CHR$ 161
```

Se tutto va bene si può eliminare il programma BASIC impartendo un NEW. Ovviamente l'UDG rimarrà in memoria, purché non si spenga il computer.

IL BOMBARDAMENTO

Il programma del bombardamento vero e proprio inizia con queste linee:

```
10 BORDER 0: PAPER 5: INK 0: CLS
20 LET a$="□□□□□□□□□□□□□
   □□□"
200 PRINT PAPER 4;AT 20,0;a$;a$;a$;a$
210 PRINT INK 1;AT 19,12;CHR$ 161; CHR$
   161;CHR$ 161
```

Queste linee visualizzano una striscia verde, alta 16 pixel lungo il margine basso dello schermo, con sopra una polveriera, e costituiscono un altro esempio dell'uso di stringhe nei lavori di grafica, oltre quelli mostrati a pagina 95.

Poiché per fare l'"erba" servono 64 blocchi colorati, la linea 20 prepara una stringa di 16 quadrati vuoti che la linea 200 usa quattro volte, a partire dall'inizio della riga 20 dello schermo fino alla 21: si evita così di dover riscrivere a$ per 64 volte!

La polveriera viene realizzata semplicemente adoperando quattro volte il carattere grafico 'r' (CHR$161), precedentemente impiegato come UDG dell'edificio, sulla strisca d'erba.

Per far entrare in azione l'aereo, servono le seguenti linee:

```
215 PAUSE 100
220 LET ay=6: LET by=ay
230 FOR x=0 TO 30
240 PRINT AT ay,x;"□";CHR$ 159
250 LET bx=x
260 IF by<19 THEN PRINT AT by+1,bx+
   1;CHR$ 160;AT by,bx;"□"
270 LET by=by+1: LET bx=bx+1
280 IF x>29 THEN PRINT AT ay,x+1;"□"
290 NEXT x
```


Anche queste sono le ormai note e consuete linee per il 'movimento dello schermo' esposte alle pagine 57-58 e ricordate in quasi tutte le sezioni di Giochi al Computer. Si noti però come, alle linee 240, 260 e 280, gli spazi occorrono per cancellare l'ultima posizione sia dell'aereo che della bomba in movimento.

IL BIG BANG

Ed eccoci alla parte principale della lezione: l'esplosione. Fare un esempio è molto più facile che descriverne il funzionamento, perciò, prima di scrivere il resto del programma, si digitino queste linee:

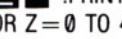
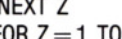
```
1000 FOR n=88 to 80 STEP-1
1010 PRINT AT 10,15; CHR$ 150
1020 POKE 23675, n
1030 PAUSE 50
1040 NEXT n
1050 STOP
```

È meglio non provare subito il programma, per non sciupare la parte già immessa. Tuttavia, scrivendo RUN1000, compare sullo schermo una lettera G, che poi svanisce gradualmente, sostituita da una F. Ciò accade perché il ciclo FOR ... NEXT diminuisce progressivamente il valore contenuto nella locazione di memoria 23675, punto di partenza degli UDG, cosicché la lettera visualizzata si modifica lentamente in quella che la precede nell'alfabeto.

Il programma dell'esplosione funziona all'incirca nello stesso modo. Per sicurezza scriviamo:

```
POKE 23675, 88
```

che riporta la locazione di memoria al suo valore originale; poi togliamo le linee da 1000 a 1050. Ora si può digitare il resto del programma per il bombardamento:

```
90 POKE 23675,88
100 FOR n= TO 31: READ a: POKE USR "a"
   +n,a: NEXT n
300 FOR e=88 TO 80 STEP-1
310 POKE 23675,e
320 PRINT INK 2;AT 19,12;CHR$ 145;
   CHR$ 147;CHR$ 145: PAUSE 6
330 PRINT INK 2; AT 19,12;CHR$ 147;
   CHR$ 145;CHR$ 147: PAUSE 6
340 NEXT e
400 POKE 23675,88
500 GOTO 210
8000 DATA 0,0,0,0,0,0,0,0,2,128,25,126,126,
   255,255,255
9000 DATA 0,0,0,0,0,0,0,0,4,33,144,66,231,
   255,255,255
```

Le linee 100, 8000 e 9000 preparano gli UDG necesari per l'esplosione. La linea 100 richiama e deposita in memoria le DATA alle linee 8000 e 9000. Le linee 300 e 340 eseguono per lo più la stessa procedura dell'esperimento precedente, togliendo un'immagine per far posto a un'altra.

C'è tuttavia una differenza importante. Gli UDG che creano l'esplosione sono quelli basati sui caratteri grafici B e D (CHR$145 e 147): quando essi spariscono lentamente dallo schermo, non ha senso che siano sostituiti da altre lettere. Perciò, le prime otto lettere di ogni frase DATA, rappresentanti i caratteri A e C (CHR$144 e 146), sono tutte 0, in modo da ottenere sullo schermo, invece di A e C che subentrano a B e D, soltanto spazi vuoti.

Per meglio osservare il processo, si inserisca una linea con PAUSE tra le linee 330 e 340.

La linea 400 è necessaria per ristabilire il valore originale (88) nella locazione di

memoria 23675, prima di riutilizzare il programma.

Programmare buoni effetti visivi sul Commodore 64 non è altrettanto facile quanto su altri home computer, dal momento che le esplosioni più convincenti si ottengono principalmente con gli sprite e con le possibilità offerte dalla grafica ad alta risoluzione. Il che significa programmi relativamente lunghi, da creare appositamente per ciascun gioco. Più avanti in questa lezione offriremo un esempio di 'esplosione' generata con sprite, utilizzabile in vari giochi. Tuttavia, esistono metodi più semplici, come questo per segmenti lampeggianti, di grande efficacia in programmi dove si richiede una qualche interruzione sullo schermo. Questo programmino può essere adottato per significare la fine di un gioco o la perdita di una 'vita'.

```
100 PRINT"♥"
110 FOR Z=0 TO 23:READ X:POKE832+Z,X
    :NEXT Z
120 SYS 832:GOTO120
130 DATA 162,0,160,200,200,208,253,160,
250,200,208,253,232
135 DATA 142,33,208,142,32,208,224,16,
    208,235,96
```

Decisamente più interessante è la prossima routine, che visualizza un disegno di rettangoli dai colori che cambiano continuamente, adottabile nelle stesse situazioni della precedente.

```
5 C$="■⊟£◣▒↑←π♠◜✕
    ○♣○✕◜♠π↑←▒◣£
    ⊟■":PRINT"♥";
8 FOR Z=0 TO 48:READ X:POKE49152+Z,X
    :NEXT Z
10 FOR Z=1 TO 24:PRINT MID$(C$,Z,1)"▚
    □□□□□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□
    □□□□";
20 NEXT Z:PRINT "⊟"
40 FOR Z=0 TO 12
60 FOR ZZ=55335-Z+(Z*40) TO 56295
    -Z-(40*Z) STEP 40:POKE ZZ,Z:NEXTZZ
70 FOR ZZ=55296+Z+(Z*40) TO 56256
    +Z-(40*Z) STEP 40:POKE ZZ,Z:
    NEXT ZZ,Z
80 DATA 169,0,141,251,0,169,216,141,252,0,
    160,0,177,251,201,0,208,2,169,16,24
82 DATA 233,0,145,251,24,230,251,165,251,
    201,0,208,2,230,252,165,251
84 DATA 201,232,208,224,165,252,201,219,
    208,218,96
100 SYS 49152:FOR Z=1 TO 250:NEXTZ:
    GOTO 100
```

La durata della routine è determinata da un ciclo FOR ... NEXT alla linea 100, che crea una pausa di 250 unità di tempo. Questo valore può venire adattato secondo le esigenze oppure esere eliminato del tutto il ciclo stesso, ottenendo un effetto ottico ipnotizzante!

Togliendo le linee 40, 60 e 70, si perdono le barre verticali, lasciando le sole strisce orizzontali di colore che si spostano verso il centro. Questa forma può essere preferibile se volessimo visualizzare un messaggio sullo schermo. La scritta andrebbe aggiunta all'istruzione PRINT della linea 20, dopo il carattere di controllo del cursore:

```
20 NEXT Z:PRINT "⊟"TAB(20)"CIAO!"
```

Per ottenere un formato adatto, si rende necessario incorporare nella linea adeguate spaziature.

ESPLOSIONE CON GLI SPRITE

Il breve programma che segue è una semplice dimostrazione dell'aspetto e del funzionamento di un tipico programma di giochi scritto in BASIC. In questo, un jet attraversa lo schermo bombardando un edificio.

```
10 PRINT "♥":POKE 53280,0:POKE53281,0
15 PRINT"◘◘◘◘◘◘◘◘◘◘
   ◘◘◘◘◘◘◘◘◘"TAB(20)"▟
   ±◢◣◘▌▌▌▌π▒▬□◘▌▌▌▌
   ▌▌▌▌†□□□"
20 FOR Z=1 TO 20:PRINT"☰"TAB(Z)"□▟
   ▒◣±▬◘▌▌▌▌□†□□"
30 POKE 1024+Z*41,32:POKE 1065+Z*41,
   42:POKE 55337+Z*41,1
40 NEXT Z:PRINT "☰"TAB(Z)"□□◘▌▌▌▌
   □□□□□"
```

Senza un'"esplosione", l'effetto di una bomba non è realistico; si aggiungano queste linee per far crollare lentamente le case:

```
45 FOR Z=1 TO 20:POKE 1884+RND(1)*2,
   RND(1)*15+110:NEXT Z
50 POKE 1885,233:POKE 1845,32:
   POKE 1884,104
```

Anche così, l'effetto non è ancora spettacolare. Ecco quindi una routine per l'esplosione basata su sprite:

```
1 FOR Z=832 TO 832+63*2:POKE Z,0:
  NEXT Z
2 FOR Z=13 TO 14:FOR A=0 TO 15:FOR B
  =0 TO 1:READ X:POKE Z*64+A*3+B,X
  :NEXT B,A,Z
3 DATA 0,0,0,0,0,0,0,0,15,240,15,140,53,92,
  54,156,54,156,53,92,15,240,15,240
4 DATA 0,0,0,0,0,0,0,0
5 DATA 15,240,15,240,53,92,53,92,213,87,
  214,151,214,151,218,167,218,167
6 DATA 214,151,214,151,213,87,53,92,53,92,
  15,240,15,240
900 V=53248:POKE V+21,3:POKE V+32,0
  :POKE V+33,0:POKE V+37,2:POKE V+
  28,3
910 POKE V+27,3:A(0)=1:A(1)=8:B(0)=4
  :B(1)=7
915 FOR LL=1TO3:IFLL=1ORLL=3THEN
  POKEV+23,0:POKEV+29,0:POKEV,183:
  POKEV+1,216:DD=0
920 IF LL=2 THEN POKEV+23,3:POKE V
  +29,3:POKE V,175:POKE V+1,208:DD=
  20
925 FOR UU=1 TO 10:FOR Z=13 TO 14:
  POKE 2040,Z:POKE V+39,B(RND(1)*2)
930 FOR T=1 TO DD:NEXT T:POKE V+38,A
  (RND(1)*2)
940 NEXT Z,UU,LL:POKE V+21,0
```


Il programma visualizza un'esplosione di colori. Una simile routine può migliorare anche programmi molto semplici.

Nonostante l'apparente complessità, il programma è piuttosto facile e, fondamentalmente, basta generare due sprite per poi scambiarli ripetutamente più volte (una spiegazione esauriente verrà data più avanti). Le DATA per gli sprite "esplosivi" sono contenute in quattro linee all'inizio del programma, precedute dalla routine di inizializzazione che libera un'area di memoria (le locazioni da 832 a 1022 fanno parte del buffer di I/O, ma si possono usare per gli sprite). Alla linea successiva, le DATA vengono lette depositate in memoria.

Si noti che Z riceve i valori di 13 e 14, i quali, moltiplicati per 64 alla linea 2, danno l'inizio della memoria di ciascun sprite (locazioni 832 e 896). Ogni sprite richiede, appunto, 64 byte.

La routine di inizializzazione deve trovarsi all'inizio di ogni programma ed ecco perché la numerazione della seconda parte del programma per l'esplosione inizia con 900, rendendola utile anche per programmi più lunghi.

La linea 900 attiva i due sprite e definisce il colore per l'area dello schermo e per la cornice. Il valore 0 (nero) alla locazione V+32 e V+33, ossia 53280 e 53281, può essere cambiato. Inoltre, viene modificato il colore del secondo "anello" di ogni sprite, per rendere l'idea di un'esplosione. L'ultima istruzione contenuta in questa linea abilita il modo multicolore per i due sprite.

Il programma passa poi ad abilitare lo sfondo (permettendo di vedere attraverso gli sprite) e a calcolare il centro dell'esplosione.

I valori numerici dopo i segni di uguale alla linea 910 si possono cambiare, ma si noterà come soltanto i colori violenti e il bianco rispondono allo scopo.

La linea seguente (915) controlla il puntatore che richiama alternativamente lo sprite piccolo o quello grande, creando l'effetto dinamico dell'esplosione. Quando i valori di LL sono 1 e 3, viene usato lo sprite piccolo, quando il valore è 2 lo sprite grande. Il ciclo FOR ... NEXT all'inizio della linea 930 aggiunge un effetto di vibrante. L'esplosione finisce alla linea 940, quando viene disabilitata la locazione dello sprite. La routine per l'esplosione può essere aggiunta ad altri programmi. Per cambiare la posizione degli sprite, si adattano i valori di V e di V+1 alle linee 915 e 920, V per la coordinata X dello schermo e V+1 per quella Y e in ogni caso la differenza tra i due è 33. Il valore di V alla linea 915 supera di 8 qualsiasi valore scelto per V alla linea 920: il valore di V+1 alle linee 920 e 915 è quindi 33 volte superiore a quello di V.

Un lampo sullo schermo è un modo molto semplice per rappresentare uno scontro in volo o nello spazio. Si trascriva e si esegua questo modulo di programma:

```
7090 MODE 1
8000 FORT T=1 TO 15
8010 FOR COL=0 TO 7
8020 VDU19,0,COL;0;
8030 NEXT COL
8040 NEXT T
8050 VDU19,0,0;0;
```

Ciò che accade è semplice: il ciclo alle linee 8000 e 8040 cambia per 15 volte i colori dello schermo, la 8010 passa a turno per gli otto colori, quindi VDU 19 riattribuisce a rotazione il colore 0, il colore dello sfondo, ad ognuno degli altri colori.

In realtà, non si notano cambiamenti di colore su tutto lo schermo: si vedono strisce di colore che salgono su di esso. Ciò accade perché i colori si susseguono così rapidamente che lo schermo non ha il tempo per colorare l'intero schermo, prima di dover cambiare colore. Ogni volta, si completa solo una piccola striscia di schermo.

Per annullare l'effetto a strisce, è sufficiente aggiungere una breve pausa che permette all'intero schermo di cambiare colore:

```
8Ø25 FOR ritardo = 1 to 2ØØ : NEXT
```

Si cambi anche la linea 8ØØØ, altrimenti il lampeggiare durerà troppo a lungo:

```
8ØØØ FOR T = 1 TO 3
```

Volendo usare il programma come subroutine in un gioco, vanno apportate due modifiche. Prima, si tolga la linea 7Ø9Ø, dal momento che il modo 1 sarà già stato definitivo nel programma principale. Secondo, si aggiunga una linea RETURN, per esempio 8Ø6Ø RETURN, al termine del programma. Se si è conservato il gioco dell'alieno (Giochi al Computer 5), adesso lo si può integrare con questo effetto. In alternativa, si usi il semplice programma di animazione qui sotto, che mostra un alieno colpito da un missile:

```
1Ø MODE1
15 VDU23;82Ø2;Ø;Ø;Ø;
17 CLS
2Ø VDU23,224,6Ø,126,219,219,126,6Ø,92,153
3Ø VDU23,225,16,16,16,56,56,56,4Ø,1Ø8
4Ø PRINTTAB(19,4)CHR$(224)
5Ø FOR Y = 31 TO 3 STEP - 1
6Ø PRINTTAB(19,Y)CHR$(225)
7Ø A = INKEY(1Ø)
8Ø PRINTTAB(19,Y)"□";
9Ø NEXT
1ØØ GOSUB 8ØØØ
11Ø GOTO 17
8Ø6Ø RETURN
```

Non si dimentichi che deve sempre esserci una linea nel programma principale che richiami la subroutine. Nel programma dell'alieno è:

```
1ØØ GOSUB 8ØØØ
```

Ogni volta che si vuole usare nei propri programmi la subroutine, si aggiungerà una simile linea (con numero di linea adeguato).

ESPLOSIONI AL SUOLO

Finora, il programma ha operato sull'intero video. Un'alternativa è realizzare una grafica animata adatta.

Ecco un'idea per l'animazione delle fiamme di un fuoco. Esse sono state concepite per essere sovrapposte a bersagli colpiti (carri armati, navi, ecc. purché siano fissi).

Questo programma per l'animazione delle fiamme è scritto come una procedura, per poter essere utilizzato in più programmi.

```
1Ø MODE 1
2Ø VDU 5
3Ø FOR T = 224 TO 229:VDU23,T
4Ø FOR P = Ø TO 7:READ A:VDU A
5Ø NEXT P
6Ø NEXT T
7Ø GCOLØ,2:MOVEØ,1ØØ:MOVE Ø,2ØØ:PLOT85,
   128Ø,1ØØ:PLOT 85,128Ø,2ØØ
8Ø Q = 1:S = 1:X = 6ØØ:Y = 232:FY = Y:BY =
   763
9Ø PROCFABBRICA
1ØØ FOR T = Ø TO 12ØØ STEP 16
11Ø MOVE T,8ØØ:GCOLØ,3:VDU229
12Ø IF T > 128 AND BY > 25Ø THEN
   PROCBOMBA
13Ø IF BY < = 25Ø AND Q < 16 THEN
   PROCESPLOSIONE
14Ø MOVE T,8ØØ:GCOLØ,Ø:VDU226
15Ø NEXT T
16Ø GOTO 8Ø
17Ø DEF PROCBOMBA
18Ø MOVE T - 16,BY:GCOLØ,Ø:VDU226
19Ø BY = BY - 16
2ØØ MOVE T,BY:GCOLØ,3:VDU228
21Ø ENDPROC
22Ø DEF PROCFABBRICA
23Ø GCOLØ,3:MOVE 6ØØ,2ØØ:MOVE692,2ØØ:
   PLOT85,6ØØ,231:PLOT85,692,231:GCOLØ,2
   :MOVE6ØØ,263
24Ø VDU227,227,227
25Ø ENDPROC
8ØØØ DEF PROCESPLOSIONE
8Ø1Ø GCOLØ,Ø:MOVE X,Y:VDU226,226,226
8Ø2Ø MOVE X,Y + 32:VDU226,226,226
8Ø3Ø GCOLØ,2:MOVE X,FY
8Ø4Ø VDU224 + S,224 + S,224 + S
8Ø5Ø S = 1 - S
8Ø6Ø FY = FY - 2
8Ø7Ø Q = Q + 1
8Ø8Ø ENDPROC
9ØØØ DATA 2,128,25,126,126,255,255,255,4,
   33,144,66,231,255,255,255
9Ø1Ø DATA 255,255,255,255,255,255,255,
   255
9Ø2Ø DATA Ø,Ø,Ø,Ø,3,15,63,255,Ø,16,16,12Ø,
   28,28,Ø,Ø,
9Ø3Ø DATA 32,16,136,254,255,8,16,32
```

Il programma mostra un aereo che lancia bombe su una fabbrica. Se questa è distrutta allora vengono stampate due serie di fiamme in rapida sequenza, per creare l'illusione di un fuoco tremolante. Dopo poco, le fiamme iniziano a estinguersi gradualmente.

Ecco come funziona il programma, linea per linea, per creare l'effetto.

Le DATA per le due serie di fiamme sono alla linea 9ØØØ e le altre frasi DATA servono per un quadrato vuoto, l'aereo e la bomba. Le linee da 3Ø a 6Ø preparano gli UDG leggendo le DATA. Le linee 8Ø3Ø e 8Ø4Ø visualizzano Le fiamme nella posizione corretta. Le linee 8Ø1Ø e 8Ø2Ø cancellano la fabbrica prima che compaiano le fiamme. Quale serie di fiamme venga adoperata dalla linea 8Ø4Ø dipende dal valore di S: inizialmente vale 1 (vedi la linea 8Ø) perciò, quando la procedura è richiamata per la prima volta, essa visualizza il carattere 244 + 1, cioè 245. La linea 8Ø5Ø cambia il valore di S in Ø, perciò al giro seguente viene stampato il carattere 244 + Ø, cioè 244. Al terzo giro, S cambia ancora in 1 e così via. Questa alternanza provoca il tremolio delle fiamme. La linea 8Ø6Ø cambia la coordinata Y delle fiamme facendole ogni volta abbassare per dare l'impressione che esse si smorzino.

Infine, la linea 8Ø7Ø tiene il conto di quante volte sono state visualizzate le fiamme e, quando Q = 16, la condizione al-

la linea 13Ø è falsa e la procedura non è richiamata fino alla prossima bomba.

Il resto del programma controlla il movimento dell'aereo e della bomba e visualizza il tutto. La linea 7Ø disegna la linea del suolo sullo schermo in basso, la 9Ø disegna la fabbrica e le linee 1ØØ, 11Ø e 14Ø spostano l'aereo in alto sullo schermo. La PROCBOMBA è richiamata dalla linea 12Ø solo se l'aereo ha raggiunto l'obiettivo nella posizione 25Ø. Quando è la bomba a raggiungere l'obiettivo, è richiamata invece la PROCESPLOSIONE dalla linea 13Ø.

Quei drammatici effetti visivi che valorizzano i giochi spaziali si ottengono sul Dragon e sul Tandy con programmi molto brevi. Si provi ad eseguire questo:

```
798Ø PMODE 3,1
799Ø PCLS
8ØØØ FOR F=1 TO 1ØØØ
8Ø1Ø SCREEN 1,Ø
8Ø2Ø SCREEN 1,1
8Ø3Ø NEXT F
8Ø4Ø CLS
```

Sullo schermo si vedono salire strisce di colore provocate dalle rapide commutazioni del computer tra i vari colori. È un effetto molto adatto per il finale di una fase particolare di un gioco.

La linea 798Ø seleziona il MODE e la 799Ø ripulisce lo schermo grafico. Le linee 8Ø1Ø e 8Ø2Ø selezionano alternativamente due colori. Il ciclo FOR ... NEXT alle linee 8ØØØ e 8Ø3Ø fa sì che ciò accada 1ØØØ volte. I colori si avvicendano molto rapidamente, tanto che in pratica l'apparecchio TV non ha mai tempo sufficiente per colorare lo schermo prima di dover cambiare di nuovo colore, cosicché appaiono soltanto delle strisce colorate.

Se il programma va usato come subroutine, o in un gioco (purché questo sia scritto nei PMODE come quello a pagina 144) o con il programma dell'alieno presentato qui sotto, le modifiche da apportare sono minime: si cancellino le linee 798Ø e 799Ø poi si aggiunga una linea RETURN numerata, ad esempio 85ØØ RETURN.

```
1Ø PMODE 1,1
2Ø DIM A(3),B(3),M(3)
2ØØ FOR K=1536 TO 2Ø16 STEP 32
21Ø READ A,B:POKE K,A:POKE K+1,B
22Ø NEXT
23Ø GET (Ø,Ø)-(15,15),A,G
24Ø GET(Ø,16)-(15,31),M,G
25Ø PCLS
26Ø MX=12Ø:MY=191:PX=12Ø:PY=2Ø
27Ø PUT(PX,PY)-(PX+15,PY+15),A,PSET
28Ø SCREEN 1,Ø
29Ø PUT (MX,MY)-(MX+15,MY+15),B,
   PSET
3ØØ MY=MY-4
31Ø IF MY<36 THEN GOSUB 8ØØØ: GOTO
   26Ø
32Ø PUT (MX,MY)-(MX+15,MY+15),M,
   PSET
33Ø GOTO 29Ø
9ØØØ DATA 252,63,3,192,15,24Ø,61,124,58,
   172,245,95,213,87,213,87
9Ø1Ø DATA Ø,128,Ø,128,Ø,128,2,16Ø,1Ø,168,Ø,
   192,3,24Ø,15,6Ø
```

Il programma visualizza un alieno colpito da un missile. A questo punto, la linea 31Ø richiama la routine dello schermo a strisce. Usato come subroutine, questo programma produrrà strisce in qualsiasi PMODE si voglia; può essere quindi aggiunto a un gioco senza cambiare le linee da 8ØØØ da 8Ø4Ø. Ecco una versione più elaborata del programma, anche se la grafica sullo schermo non resta intatta come nel caso precedente. Lo si può scrivere ed eseguire così com'è, oppure aggiungerlo come subroutine a un altro programma, tipo l'animazione dell'alieno. In tal caso, occorre cancellare la linea 799Ø e aggiungere una linea RETURN, come già descritto.

```
799Ø PMODE 3,1
8ØØØ FOR F=1 TO 3
8Ø1Ø FOR K=Ø TO 1
8Ø2Ø SCREEN 1,K
8Ø3Ø FOR J=1 TO 4
8Ø4Ø PCLS J
8Ø5Ø NEXT J
8Ø6Ø NEXT K
8Ø7Ø NEXT F
8Ø8Ø CLS
```

Come subroutine funziona sia nei modi a due colori (PMODE Ø,2 e 4) che in quelli a quattro (PMODE 1 e 3). Oltre a cambiare la colorazione, la routine ripulisce anche lo schermo, cambiandone il colore.

Le linee da 8Ø3Ø a 8Ø5Ø contengono un ciclo FOR ... NEXT che riporta lo schermo in ognuno dei colori tra quelli a disposizione. Il programma prevede la ricerca di quattro colori alla volta, ma funziona correttamente anche nei modi a due colori senza alcun inconveniente.

Un'ultima variante al programma è di questo tipo:

```
799Ø PMODE 3,1
8ØØØ FOR F=1 TO 5
8Ø1Ø SCREEN 1,Ø
8Ø2Ø FOR K=1 TO 2ØØ: NEXT K
8Ø3Ø SCREEN 1,1
8Ø4Ø FOR K=1 TO 2ØØ: NEXT K
8Ø5Ø NEXT F
8Ø6Ø CLS
```

Anch'essa può venir provata separatamente o come subroutine (cancellando la linea 799Ø e aggiungendo una linea RETURN).

Lo schermo lampeggerà in colori diversi. Ciò avviene perché le linee 8Ø2Ø e 8Ø4Ø introducono pause sufficienti a colorare completamente lo schermo prima dello scambio tra i set di colore. Altri esperimenti possono riguardare la lunghezza delle pause e il numero di volte in cui i colori devono scambiarsi.

FIAMME

Finora, gli effetti considerati hanno trattato lo schermo nella sua interezza, ma si possono ottenere effetti visivi migliori

con un po' di grafica d'animazione.

L'idea che segue crea l'animazione di alcune fiamme da sovrapporre a un bersaglio colpito e può venire adoperata in ogni gioco che contempli bersagli fissi.

Il programma è scritto nel PMODE 1 e non può essere usato, senza modificarlo, in giochi scritti in altri PMODE. Si trascriva e lo si esegua con un RUN:

```
10 PMODE 1,1
20 DIM B(3),E1(3),E2(3)
30 FOR K=1536 TO 2016 STEP 32
40 READ A,B:POKE K,A:POKE K+1,B
50 NEXT
60 GET (0,0)-(15,15),E1,G
70 GET (0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS:HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT (HX,HY+N)-(HX+15,HY+15),
   E1,PSET
8020 FOR K=1 TO 100:NEXT
8030 PUT (HX,HY+N)-(HX+15,HY+15),
   E2,PSET
8040 FOR K=1 TO 100:NEXT
8050 PUT (HX,HY+N)-(HX,+15,HY+15),
   B,PSET
8060 NEXT
9000 DATA 0,12,192,0,3,195,63,252,63,252,
   255,255,255,255,255,255
9010 DATA 0,48,12,3,195,0,48,12,252,63,255,
   255,255,255,255,255,255
```

Sullo schermo appaiono alternativamente due serie di fiamme, per creare la giusta illusione di un fuoco, che gradualmente si smorza.

Le DATA per le fiamme si trovano alle linee 9000 e 9010. Le linee dalla 30 alla 50 visualizzano le fiamme sullo schermo.

Poiché il programma prevede un modo a quattro colori, le DATA sono un po' diverse da quelle viste in Codice Macchina 2 (pagina 40). Vedremo la procedura per UDG a colori più avanti.

Le linee dalla 60 alla 70 inseriscono le forme delle due fiamme nelle matrici E1 e E2, DIMensionate nella linea 20.

Le linee da 8000 a 8060 creano l'animazione delle fiamme e il loro spegnimento. Ogni volta che il programma attraversa il ciclo FOR ... NEXT le due matrici con le fiamme vengono riprodotte sullo schermo e cancellate "coprendole" con una matrice vuota B. La punta delle fiamme è abbassata ogni volta dal +N nelle linee di PUT per creare l'effetto di smorzamento.

Per meglio comprendere il funzionamento, si facciano visualizzare le coordinate delle linee di PUT mentre il ciclo sta girando.

Il programma funzionerebbe anche da solo, ma è certamente più utile come subroutine del programma di bombardamento qui sotto, in cui un edificio viene distrutto e bruciato. Associando i due programmi, va cambiata la linea 20 in modo che riguardi anche il bombardamento. Va anche cancellata la linea 80, scrivendo 80 ENTER.

```
20 DIM A(3),B(3),H(3),E1(3),E2(3)
200 FOR K=1536 TO 2016 STEP 32
210 READ A,B:POKE K,A:POKE K+1,B
220 NEXT
230 GET(0,0)-(15,15),A,G
240 GET(0,16)-(15,31),H,G
250 PCLS:LINE(0,163)-(255,191),PSET,BF
260 HX=124:HY=146:PX=0:PY=40:B
   =0
270 PUT(HX,HY)-(HX+15,HY+15),B,PSET
280 SCREEN 1,0
290 PUT(PX,PY)-(PX+15,PY+15),B,PSET
300 PX=PX+4
310 PUT(PX,PY)-(PX+15,PY+15),A,PSET
320 IF PX=20 THEN B=1:BX=PX+8:BY
   =PY+8
330 IF B=1 THEN PRESET(BX,BY):PRESET
   (BX+2,BY):BX=BX+2:BY=BY+2:
   PSET(BX,BY,4):PSET(BX+2,BY,4)
340 IF BY=148 THEN GOSUB 8000: BY=0:
   GOTO 250
350 GOTO 290
8070 RETURN
```

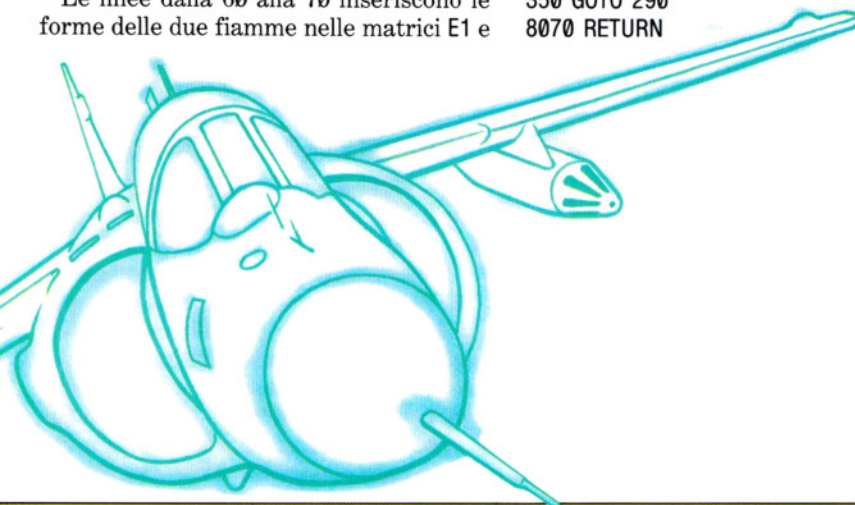
```
9020 DATA 0,0,2,0,130,128,162,160,170,170,
   162,160,130,128,2,0
9030 DATA 0,3,12,51,60,243,255,255,255,
   255,85,85,86,149,86,149
```

Occorrono alcune modifiche se la subroutine dovesse essere usata in altri PMODE. Ecco due versioni che permetteranno alle fiamme di adattarsi a giochi scritti in altri modi.

Per primo, un programma adatto ai PMODE 3 e 4 (si cambi il numero del modo correttamente):

```
10 PMODE 3,1
20 DIM B(6),E1(6),E2(6)
30 FOR K=1536 TO 2496 STEP 64
40 READ A,B:POKE K,A:POKE K+1,B
45 POKE K+32,A:POKE K+33,8
50 NEXT
60 GET(0,0)-(15,15),E1,G
70 GET(0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS: HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT(HX,HY+N)-(HX+15,HY+15),
   E1,PSET
8020 FOR K=1 TO 100:NEXT
8030 PUT(HX,HY+N)-(HX+15,HY+15),
   E2,PSET
8040 FOR K=1 TO 100:NEXT
8050 PUT(HX,HY+N)-(HX+15,HY+15),B,
   PSET
8060 NEXT
9000 DATA 0,12,192,0,3,195,63,252,63,252,
   255,255,255,255,255,255
9010 DATA 0,48,12,3,195,0,48,12,252,63,255,
   255,255,255,255,255
```

Per secondo, un programma adatto al PMODE 2:

```
10 PMODE 2,1
20 DIM B(6),E1(6),E2(6)
30 FOR K=1536 TO 2496 STEP 32
40 READ A:POKE K,A:POKE K+16,A
50 NEXT
60 GET(0,0)-(15,15),E1,G
70 GET(0,16)-(15,31),E2,G
80 SCREEN 1,0
250 PCLS: HX=124:HY=146
8000 FOR N=0 TO 15
8010 PUT(HX,HY+N)-(HX+15,HY+15),
   E1,PSET
8020 FOR K=1 TO 100:NEXT
8030 PUT(HX,HY+N)-(HX+15,HY+15),
   E2,PSET
8040 FOR K=1 TO 100:NEXT
8050 PUT(HX,HY+N)-(HX+15,HY+15),B,
   PSET
8060 NEXT
9000 DATA 2,128,25,126,126,255,255,255
9010 DATA 4,33,144,66,231,255,255,255
```

GLI SPRITE SUL COMMODORE 64

Lo sprite è una caratteristica standard nel Commodore 64, che semplifica la gestione grafica ad alta risoluzione. Facile da controllare, sta alla base di molti giochi

Lo sprite, chiamato anche MOB (da Movable Object Block), è un tipo speciale di UDG ad alta risoluzione molto mobile, che in parte abbiamo già visto all'opera. Oltre alla mobilità, sue caratteristiche non comuni sono la capacità di espandersi o contrarsi a comando in larghezza e in altezza. Non c'è da stupirsi quindi del largo uso di sprite nella programmazine di giochi, sebbene essi risultino utili ovunque sia richiesta una grafica animata ad alta risoluzione. Si possono usare sprite, per esempio, anche in programmi commerciali, per creare simboli iconici (indicativi di una scelta sulle operazioni del programma) oppure come parte di una rappresentazione grafica.

Ci sono due tipi di sprite, la forma standard ad alta risoluzione e quella multicolore. La differenza tra i due è che il primo adotta il colore scelto per il primo piano. Gli sprite multicolore, invece, offrono una scelta fino a quattro colori insieme, ma con una certa perdita di risoluzione orizzontale. Per cominciare, occupiamoci degli sprite standard: quelli multicolore verranno trattati a parte.

DEFINIRE UNO SPRITE

La definizione degli sprite è molto simile a quella dei normali UDG (vedere a pagina 38), ma i primi sono più grandi e richiedono più DATA. Per l'esattezza, uno sprite ha tre volte la larghezza di un UDG standard e quasi tre volte l'altezza, occupando quindi un'area di 24 pixel per 21. Non tutti i 5Ø4 pixel disponibili vengono usati (ossia "accesi") e in alcuni casi ciò non sarebbe possibile.

Invece di definire ognuno dei 24 pixel sepratamente su ogni riga, l'informazione sulla forma dello sprite è contenuta in tre gruppi di otto bit (tre byte) per ciascuna delle 21 righe. Questo offre un criterio per adattare agli sprite i valori delle frasi DATA usati per i caratteri UDG, dal momento che ambedue risultano calcolabili in gruppi di otto. La disposizione dei byte nello sprite prende la forma seguente:

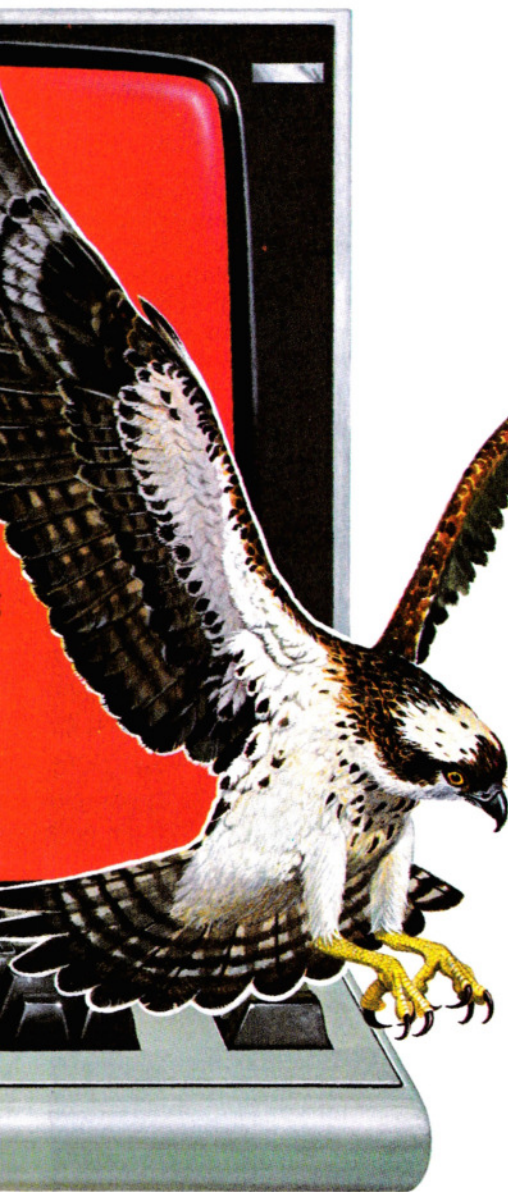
```
Row 1:   BYTE 1   BYTE 2   BYTE 3
Row 2:   BYTE 4   BYTE 5   BYTE 6
Row 3:   BYTE 7   BYTE 8   BYTE 9
```

... proseguendo fino a:

```
Row 2Ø:  BYTE 58  BYTE 59  BYTE 6Ø
Row 21:  BYTE 61  BYTE 62  BYTE 63
```

Si hanno così dei raggruppamenti di tre byte per ognuna delle 21 righe dello sprite. Ognuno di questi byte tratta informazioni allo stesso modo di una singola linea di UDG (vedere a pagina 38). In termini di notazione decimale, per esempio, i valori possibili per le posizioni dei pixel su ciascuna riga sono mostrati nel diagramma a pagina 17Ø.

Se tutte le otto posizioni dei pixel disponibili in ogni byte vengono usate (accese o abilitate) il valore decimale corrispondente di quel byte sarebbe 128+64+32+16+8+4+2+1, cioè 255. Se non si usasse nessun pixel, la somma dei valori binari darebbe Ø. Gli unici valori corrispondenti a ogni possibile conversione dei pi-

- COS'È UNO SPRITE
- DEFINIRE UNO SPRITE
- PRIMI PASSI
- LA GENERAZIONE DELLE **DATA**
- USARE LO SPRITE
- SPRITE IN MEMORIA
- UN SEMPLICE TRUCCO PER TRACCIARE DISEGNI
- COME EVITARE I PROBLEMI DI **SAVE** E **LOAD**

xel 'usati' stanno tra questi due estremi. Il valore ottenuto può rientrare direttamente in una frase DATA per concorrere nella definizione di uno sprite.

PRIMI PASSI

Vediamo ora alcuni esempi. Come con i comuni UDG, il miglior modo per cominciare è disegnare la forma voluta su un foglio quadrettato. Si evidenzi il perimetro di un rettangolo di 24 per 21 quadretti e si indichino le divisioni dei byte (oppure si fotocopi la griglia riprodotta qui o sul manuale). Nella pagina successiva è riportato un esempio di griglia, adoperata nel programma della stazione spaziale a pagina 151.

In esso vengono elencati i valori delle DATA, ai quali è bene prestare un po' d'attenzione. Il primo byte nella prima riga non è usato, così il valore della DATA di quel pixel è Ø. Nel byte successivo, sono usati tutti i pixel meno uno, che occupa l'ultimo posto. Il valore totale è perciò 128+64+32+16+8+4+2=254. Il terzo byte nella prima riga è inutilizzato, fornendo così altri Ø alla DATA. I valori della riga 1 sono perciò Ø,8,Ø.

Prima di procedere, osserviamo il resto della disposizione dei pixel nello sprite. È facile trovare un'altra riga che abbia la stessa configurazione per uno o più byte, specie in disegni simmetrici. In questi casi si evita il calcolo delle DATA: per la riga 1 non ci sono repliche, ma ciò avviene per la 14 e la 16 oppure per la 17 e la 19.

Passiamo alla riga successiva dove, nel primo byte, sono accesi i pixel 2 e 1, per un valore di 2+1=3.

Nel successivo byte, sono accesi i pixel 6,5,4 e 1, per un valore di 32+16+8+1=57. Nell'ultimo byte, viene usato solo il primo pixel (128). Il totale della riga risulta perciò 3, 57, 128.

Nella riga 3, il primo byte ha tre pixel attivati e un totale di 4+2+1=7. Il secondo byte ha tutti i pixel accesi, per un valore di 255. L'ultimo byte ha 168+64=192.

Nello stesso modo, si proceda a calcolare i valori delle altre righe, dopodiché si possono raggruppare i valori di tutte le righe per le frasi DATA relative allo sprite della stazione spaziale:

```
20 DATA Ø,254,Ø,3,57,128,7,255,192,Ø,16,Ø,
   16,56,16,56,84,56,124,148,124,131,255
30 DATA 13Ø,144,58,18,184,16,58,144,16,18,
   131,255,13Ø,254,84,254,252,56,126,Ø,56
40 DATA Ø,Ø,4Ø,Ø,Ø,56,Ø,1,199,Ø,6,16,192,1,
   199,Ø,Ø,124,Ø
```

Questo sprite avrà l'aspetto di quello nel programma di pagina 151. Non sempre occorre scriverlo per esteso in questo modo: con opportuni adattamenti, purché non si crei un vuoto nei valori, si possono di solito omettere le righe iniziale e finale, se inutilizzate, nella definizione di uno sprite.

Si esamini ora il programma della stazione spaziale per ricostruire i valori usati e riconoscerli nelle frasi DATA delle linee 2Ø e 14Ø.

PROGRAMMA PER GENERARE DATA

La parte meno entusiasmante nell'uso di sprite è rappresentato dal calcolo dei 63 valori per le frasi DATA necessari a definirle. Invece di compiere manualmente questa operazione, si può acquistare uno dei tanti programmi di utilità in commercio che calcoli i valori necessari.

Ma, per iniziare, basta un programma relativamente semplice che consenta di disegnare lo sprite sullo schermo per modificarlo a piacimento.

Successivamente, il medesimo programma calcolerà i valori corrispondenti delle DATA da usare per definire lo sprite in altri programmi.

Il programma che segue ha questo scopo e ai possessori di una stampante offre anche l'opzione di ottenere una copia stampata come riferimento. Il programma è adatto solo per sprite ad alta risoluzione con un colore, tuttavia è pur sempre molto utile. Se la trascrizione può apparire sproporzionata rispetto al lavoro di calcolo necessario per un solo sprite, si pensi che, una volta che è stata eseguita, il programma può essere memorizzato e riutilizzato per ogni successiva definizione di sprite:

```
10 POKE 5328Ø,1:POKE 53281,1:DIMA$(21),Z
   (3,21),A(24)
20 PRINT "♡▓"TAB(13)"SPRITE EDITOR
   ▟▟"
30 PRINT TAB(8);:INPUT"←
   STAMPANTE ATTIVA(↑S/N←)↑";I$:
   IF I$="S"THEN PR$="S"
40 IF I$<>"S" AND I$<>"N" THEN
   GOTO 2Ø
50 PRINT"♡±"TAB(13)"ATTENDERE"
60 FOR Z=1 TO 8:READA(Z):A(Z+8)=A(Z):
```

Un esempio di uno sprite per il gioco della stazione spaziale: la stazione stessa, sovrapposta alla griglia con i numeri delle righe e dei bit. I valori dei bit si sommano per formare i valori per le DATA. Ci sono tre numeri per ogni riga in quanto costituita da tre byte di otto pixel ciascuna. La griglia può essere ricalcata o fotocopiata per progettare i propri sprite e calcolare i valori delle DATA

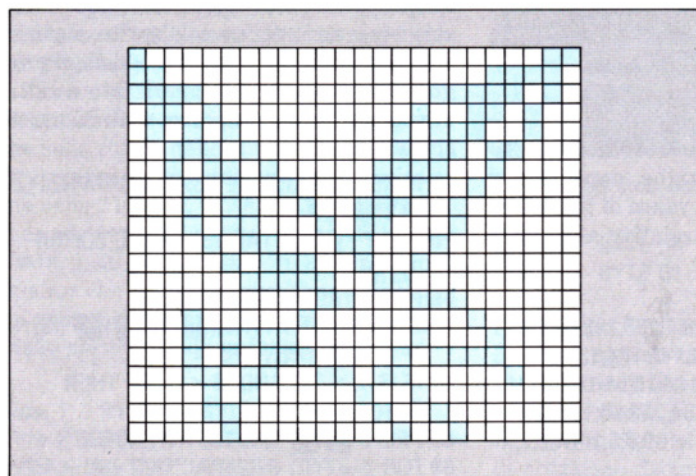
```
250 DATA"
260 DATA"
270 DATA"
280 DATA"
290 DATA"
300 DATA"
310 DATA"
320 DATA"
330 DATA"
340 DATA"
350 DATA"
360 DATA"
370 DATA"
380 DATA"
390 DATA"
400 DATA"
410 DATA"
420 DATA"
430 DATA"
440 DATA"
450 DATA"
```

```
    A(16+Z)=A(Z):NEXT:DATA128,64,32,16,
    8,4,2,1
70 FOR Z=1 TO 21:READ A$(Z)
80 FOR ZZ=1 TO 8:IF MID$(A$(Z),ZZ,1)=
    "*"THEN Z(1,Z)=Z(1,Z)+A(ZZ)
90 NEXT ZZ:FOR ZZ=9 TO 16:IF MID$(A$
    (Z),ZZ,1)="*" THEN Z(2,Z)=Z(2,Z)+A
    (ZZ)
100 NEXT ZZ:FOR ZZ=17 TO 24:IF MID$(A$
    (Z),ZZ,1)="*" THEN Z(3,Z)=Z(3,Z)+A
    (ZZ)
110 NEXT ZZ,Z:PRINT "■■":IF PR$=
    "S" THEN OPEN4,4:CMD4
120 PRINT TAB(12)"■DATI DEI
    CARATTERI■■"
130 FOR Z=1 TO 21:PRINT Z(1,Z);",";Z(2,Z);
    ",";Z(3,Z);:IF Z<21 THEN PRINT",";
140 NEXT Z:PRINT:PRINT:IF PR$="S"
    THEN GOTO 170
150 PRINT TAB(7)"■■PREMERE RETURN
    PER CONTINUARE"
160 GET K$:IF K$<>CHR$(13) THEN GOTO
    160
170 PRINT "■"TAB(11)"■CREAZIONE DEI
    CARATTERI■■"
180 PRINT TAB(7)"■76543210765432107
    6543210"
190 FOR Z=1 TO 21:PRINT TAB(7)"■";
200 FOR ZZ=1 TO 24:IF MID$(A$(Z),ZZ,1)=
    "*" THEN PRINT "■□■";:GOTO 220
210 PRINT "."
220 NEXT ZZ:PRINT"■";Z:NEXT Z:IF PR$=
    "S" THEN PRINT #4, "□":CLOSE4
230 IF PR$<>"S" THEN GOTO 230
240 REM□□765432107654321076543210
```

Prima si disegni l'uccello in volo sulla carta quadrettata. Poi, mediante asterischi, lo si immetta nel programma per generare DATA

Per adoperare il programma, si usi l'istruzione LIST 240- che visualizza le 21 linee in fondo la programma, tutte linee di DATA vuote, in cui disegnare la forma dello sprite da progettare. La posizione di ogni carattere nella frase DATA rappresenta una posizione valida dei pixel: la linea è larga 24 caratteri. Con il controllo cursore ci si sposta lungo il reticolo e si pone un asterisco in ogni punto del disego, ricordandosi di premere RETURN per immettere ogni linea.

Per comodità, la frase REM alla linea 240 numera ognuna delle 24 posizioni orizzontali disponibili per ciascuna delle 21 righe.

In basso a sinistra, viene mostrato un esempio tipico, nel quale si vede il disegno creato con degli asterischi e usato per definire la figura di un uccello in volo.

Quando il disegno è completato, si im-

partisca un RUN: sullo schermo comparirà un messaggio in cui si chiede se adoperare la stampante. Premendo [N], la visualizzazione avviene sullo schermo e i numeri che appariranno sono i primi 63 numeri da ricopiare e da usare della frase DATA. Nell'esempio mostrato, la frase DATA risulterà così:

```
DATI DEL CARATTERE 128,0,3,192,0,30,240,
    0,250,104,1,52,84,2,228,58,2,216,45,7,
    144,20,133,32,22,78,64,22,44,64,11,24,
    128,5,201,0,3,230,0,0,49,192,16,64,32,31,
    129,16,21,192,240,27,54,8,10,9,0,12,4,
    128,4,2,64
```

Premendo [RETURN], si ottiene una visualizzazione su grande scala dello sprite. Se volessimo una copia del tutto su stampante, occorre premere contemporaneamente [RUN/STOP] e [RESTORE] e dare un nuovo RUN. Alla domanda concernente la stampa si risponda S: si otterrà prima un listato di tutte le DATA, poi una stampa ingrandita dello sprite sulla griglia di costruzione.

Ogni sprite definito con questo programma può essere memorizzato come parte dello stesso programma.

In seguito, lo si può usare nuovamente per apportarvi modifiche o per creare nuovi sprite.

USARE LO SPRITE

Il progetto e la definizione delle DATA sono forse la parte più facile nell'uso di uno sprite. I valori delle DATA non hanno, di per sé, alcuna utilità: affinché lo sprite serva a qualcosa, occorre incorporare i valori in un programma.

Per prima cosa è necessario memorizzare la definizione dello sprite per poterla richiamare e inserire nel programma scelto. Spiegheremo più avanti come far questo. Per il momento si scriva questo programma e lo si lanci:

```
10 V=53248:X=150:Y=157:PRINT "♥"
20 FOR I=16000 TO 16062:READ A:POKEI,A
   :NEXT I
25 POKE 2040,250:POKE V+21,1:GOTO 50
30 GET A$:A=0:XX=0:IF A$="P" THEN A
   =1:GOTO 50
35 IF A$="L" THEN A=2:GOTO 50
40 IF A$="Z" THEN XX=-2
45 IF A$="X" THEN XX=+2
30 FOR Z=1 TO 10:X=X+XX:IF X>
   250 THEN X=30
55 IF X<20 THEN X=250
60 IF A=1 AND Y>70 THEN Y=Y-2
65 IF A=2 AND Y<200 THEN Y=Y+2
70 POKE V,X:POKE V+1,Y
75 NEXT Z:GOTO 30
100 DATA 128,0,3,192,0,30,240,0,250,104,1,
    52,84,2,228,58,2,216,45,7,144,20
105 DATA 133,32,22,78,64,22,44,64,11,24,
    128,5,201,0,3,230,0,0,49,192,16,64
110 DATA 32,31,129,16,21,194,240,27,54,8,
    10,9,0,12,4,128,4,2,64
```

Il programma definisce uno sprite e lo muove sullo schermo, sotto il controllo dei tasti [L], [P], [Z], e [X].

In questo caso, lo sprite è la figura dell'uccello in volo vista precedentemente, ma si può usare il medesimo programma per qualsiasi sprite da definire a un solo colore: basta intervenire opportunamente sulle frasi DATA.

Esaminando le linee 1ØØ, 1Ø5 e 11Ø, si noterà che queste elencano i valori delle DATA dell'uccello, generati dal programma di utilità.

La linea 1Ø prepara una serie di variabili, la base per realizzare uno sprite, e ripulisce lo schermo.

La linea 2Ø legge le DATA che definiscono lo sprite e le conserva in memoria. La linea 3Ø abilita o accende l'immagine. Le linee dalla 5Ø alla 75 istruiscono il computer a muovere lo sprite sotto il controllo della tastiera.

Per visualizzare lo sprite si provino a eliminare le linee 4Ø, 5Ø, 55, 6Ø, 7Ø e 75 dal programma e l'uccello apparirà in un punto, determinato dalle variabili X e Y, poste a 15Ø e 157 (all'incirca nel mezzo dello schermo).

Assegnando altri valori alla X e alla Y (linea 1Ø), l'uccello può essere spostato in altri punti dello schermo.

La parte più complessa, per il momento, va dalla linea 1Ø alla 25 e riguarda la memorizzazione dello sprite: tra breve questa verrà trattata nei particolari. Una delle prossime lezioni è dedicata a una approfondita spiegazione sul funzionamento della memoria.

Nell'usare il programma per visualizzare i numeri per lo sprite, c'è un modo rapido di passare dal progetto alle frasi DATA del programma.

Usando la carta con il progetto, se ne sottolineino le righe chiave (la dimensione dovrebbe essere a misura dello schermo) e poi si attacchi il foglio allo schermo, dopo aver LISTato le linee 24Ø-45Ø. Ora, con il cursore e gli asterischi, ripassiamo le linee segnate fino a farle completamente combaciare con il progetto sul foglio.

Aggiungendo queste ulteriori linee al programma descritto nel testo, si avrà un semplice gioco di caccia

```
15 POKE 53280,2:POKE 53281,2:
   POKE 650,128:FOR Z=
   16000 TO 16000+64*2:POKE,0:NEXT
20 FOR I=16000 TO 16077:READ A:
   POKE I,A:NEXT I:TI$="000000":
   POKE V+29,1
25 POKE 2040,250:POKE 2041,251:
   POKE V+21,3:XX=100:YY=100:
   POKE V+40,1
30 FOR Z=1 TO 5:PRINT "[HOME][DOWN]TEMPO:
   [RIGHT]"VAL(TI$):IF RND(1)>
   .30 THEN POKE V+23,RND(1)*2
35 A=INT(RND(1)*3)+1:X=X+10:IF X
   >239 THEN X=30
40 IF A=1 AND Y>70 THEN Y=Y -10
45 IF A=2 AND Y<200 THEN Y=Y+10
50 POKE V,X:POKE V+1,Y:POKE V+
   39,7:GET Z$:IF Z$="Z" AND XX>
   30 THEN XX=XX-5
55 IF Z$="X" AND XX<250 THEN XX
   =XX+5
60 IF Z$="P" AND YY>50 THEN YY
   =YY-5
65 IF Z$="L" AND YY<220 THEN YY
   =YY+5
70 POKE V+2,XX:POKE V+3,YY:
   IF PEEK(V+30)=3 THEN S=S+1
75 PRINT"[HOME][DOWN]"TAB(25)"PUNTI:[RIGHT]"S:
   NEXT Z:IF VAL(TI$)<59 THEN 30
80 PRINT "[HOME][■][▒] TEMPO SCADUTO
   [DOWN]!": END
115 DATA 8,0,0,8,0,0,62,0,0,8,0,0,8,0,0,
```

SPRITE IN MEMORIA

La definizione di uno sprite richiede 63 byte di memoria per i 63 valori dalle DATA, ma, per semplificare alcuni calcoli, conviene allocare 64 byte per ciascuna definizione. Questi 64 byte possono essere immagazzinati in una qualsiasi area di memoria libera, purché sia un multiplo di 64. Ogni definizione possiede un'indirizzo e si usano a questo scopo gli speciali puntantori degli sprite. Ne esistono otto e a ognuno può essere assegnato un valore da Ø a 255. Questo valore viene moltiplicato per 64, per individuare la locazione dello sprite. (Ecco perché la locazione di memoria deve essere un multiplo di 64).

Il valore massimo, 255, dà quindi un valore limite di 255 × 64, cioè 16K, un intero blocco di memoria a cui può accedere un chip del video (I blocchi o "banchi" di 16K sono quattro, accessibili uno per volta).

I puntatori sono un mezzo efficace per abilitare o disabilitare qualsiasi definizione di sprite presente in memoria. Si può così richiamare un'intera sequenza di sprite, ottenendo fantasiosi effetti di animazione. È questo, in pratica, il modo più comodo di usare gli sprite: commutare i puntatori anziché gli stessi sprite, lasciandoli disponibili per altri usi.

I puntatori sono sempre collocati negli ultimi otto byte inutilizzati della memoria dello schermo: in genere, da 2Ø4Ø a 2Ø47. Un esempio di ciò è alla linea 3Ø del programma dell'uccello, dove si usa 2Ø4Ø.

La memoria dello schermo si può riallocare, spostando però anche i puntatori. I loro nuovi valori andranno depositati, mediante POKE, nelle nuove locazioni.

LE SAVE SU NASTRO

Un inconveniente meno raro di quello che si creda, lavorando con gli sprite, è quello relativo alla memorizzazione su nastro dei programmi. Il problema sorge se si è eseguito un RUN (succede!) nel corso dello sviluppo di un programma che conteneva uno sprite. Può sembrare normale, ma troppo spesso si dà la colpa di questi errori alla unità a cassette C2N! La sola soluzione infallibile è far precedere il comando SAVE dalla seguente POKE:

```
POKE 53269,Ø:SAVE "NOMEPROG"
```

Con ciò si disattiva la visualizzazione dello sprite e si elimina la fonte di interferenza che disturba la procedura di memorizzazione su nastro. In alternativa, si aggiunga la POKE alla fine del programma.

La stessa POKE dovrebbe precedere un'istruzione LOAD quando si carica sul Commodore un nuovo programma dopo che si è visualizzato uno sprite. Ovviamente, ciò e superfluo se si spegne e si riaccende l'apparecchio!

Chi usa dischetti sarà felice di sapere che i problemi con SAVE e LOAD connessi con gli sprite non investono queste unità e si possono usare le procedure consuete.

IL CENTRO DI COMANDO

Per controllare gli sprite, si devono fare i conti con il funzionamento del chip 6566 VIC-II del Commodore 64. In particolare, si deve saper accedere ai comandi di controllo. La cosa non è difficile come sembra e verrà spiegata dettagliatamente in una delle prossime lezioni. Per adesso, si osservi la tabella a lato, che elenca 47 indirizzi di memoria (da 53.248 a 53.294) usati nella programmazione degli sprite. Il comunemente usato "V + valore" mostra il rapporto tra tutti gli indirizzi correlati, basato sull'indirizzo di partenza, dove V = 53248.

La notazione "V + valore" si ricorda più facilmente nell'indirizzo specifico ed è un impiego più funzionale della memoria, specialmente nei programmi più lunghi.

Inoltre, questo metodo identifica in modo chiaro le specifiche istruzioni, il che è utile nelle correzioni e nelle modifiche dei programmi.

Si osservi che ciascuno degli otto sprite normali è numerato con un valore che ha importanti conseguenze.

Dalla tabella si rileva anche che le locazioni prevedono il controllo di tutto: dalla posizione degli sprite, alla individuazione di collisioni, al colore.

L'uso della notazione "V + valore" è descritto con maggiore approfondimento nella seconda parte di questa lezione, nella quale si espone il funzionamento interno del controllo degli sprite. Ci si può già togliere un po' di curiosità, considerando l'uso di V + valori nel programma dell'uccello.

Alla linea 25 c'è il V + 21, il comando di abilitazione che accende lo sprite. Alla linea 7Ø i valori V definiscono le posizioni X e Y dello sprite sullo schermo. Altri esempi da analizzare sulla tabella compaiono alla linea 24ØØ e altri nel programma della stazione spaziale.

Locazioni di memoria del chip VIC-II

Questa tabella di riferimento è molto utile per accedere alle locazioni di memoria usate nel controllo di forma, dimensioni, colore e posizione degli sprite. Il formato V + è il più comodo da ricordare.

Decimale	Val. V +	Descrizione
53248	V	Posizione X Sprite Ø
53249	V + 1	Posizione Y Sprite Ø
53250	V + 2	Posizione X Sprite 1
53251	V + 3	Posizione Y Sprite 1
53252	V + 4	Posizione X Sprite 2
53253	V + 5	Posizione Y Sprite 2
53254	V + 6	Posizione X Sprite 3
53255	V + 7	Posizione Y Sprite 3
53256	V + 8	Posizione X Sprite 4
53257	V + 9	Posizione Y Sprite 4
53258	V + 10	Posizione X Sprite 5
53259	V + 11	Posizione Y Sprite 5
53260	V + 12	Posizione X Sprite 6
53261	V + 13	Posizione Y Sprite 6
53262	V + 14	Posizione X Sprite 7
53263	V + 15	Posizione Y Sprite 7
53264	V + 16	Byte alto, coord. X
53265	V + 17	Registro contr. VIC
53266	V + 18	Registro scansione
53267		(penna ottica)
53268		(penna ottica)
53269	V + 21	Abilitazione sprite
53270	V + 22	Registro contr. VIC
53271	V + 23	Espansione Sprite Ø-7 Y
53272	V + 24	Contr. memoria VIC
53273	V + 25	Registro d'interrupt
53274	V + 26	Abilitazione interrupt
53275	V + 27	Priorità dello sfondo
53276	V + 28	Selezione multicolore
53277	V + 29	Espansione Sprite Ø-7 X
53278	V + 30	Rivelatore collisione (sprite)
53279	V + 31	Rivelatore collisione (sfondo)
53280	V + 32	Colore margine schermo
53281	V + 33	Colore sfondo Ø
53282	V + 34	Colore sfondo 1
53283	V + 35	Colore sfondo 2
53284	V + 36	Colore sfondo 3
53285	V + 37	Sprite multicolore 1
53286	V + 38	Sprite multicolore 2
53287	V + 39	Colore Sprite Ø
53288	V + 40	Colore Sprite 1
53289	V + 41	Colore Sprite 2
53290	V + 42	Colore Sprite 3
53291	V + 43	Colore Sprite 4
53292	V + 44	Colore Sprite 5
53293	V + 45	Colore Sprite 6
53294	V + 46	Colore Sprite 7

UN PO' DI FORMA NEI PROGRAMMI

- COSA SIGNIFICA PROGRAMMAZIONE STRUTTURATA
- L'USO DEI DIAGRAMMI A BLOCCHI
- L'USO DI STRUTTURE NEI PROGRAMMI **BASIC**

Aver metodo nella progettazione facilita la comprensione dei programmi e il lavoro su di essi. La differenza tra un buon risultato e un fallimento consiste anche in questo

Quando ci si accinge a scrivere il primo programma per un computer, la fretta di mettersi alla tastiera e partire subito a scrivere un pezzo di programma è di solito incontrollabile. Può anche darsi che le prime linee vadano bene. Perciò, ne aggiungiamo altre e anche queste funzionano e allora ne continuiamo ad aggiungere qua e là, verificando man mano il programma, che intanto è cresciuto fino a un centinaio di linee.

Poi all'improvviso, il disastro! Ancora qualche linea e non funziona più niente. Non c'è un motivo visibile per cui abbia dovuto fermarsi. Cambiamo una linea, un'altra: niente.

Comunque non arrendiamoci: ci sono metodi perché ciò non accada e serve solo un po' di organizzazione. Se seguiamo alcune semplici regole nella scrittura dei programmi, non dovremmo più avere problemi.

PROGRAMMAZIONE STRUTTURATA

Il BASIC standard è stato inventato per essere facile da usare, ma a confronto con altri linguaggi, possiede pochissime strutture e ciò ostacola la scrittura di programmi ben strutturati. Le strutture sono i blocchi di costruzioni da usare per dare forma la programma. In BASIC standard sono: IF ... THEN, FOR ... NEXT, GOTO e GOSUB e nel BASIC BBC anche alcune altre: REPEAT ... UNTIL, PROCedure e funzioni. La maggior parte di queste sono di per sé comprensibili, ma l'idea ora è di raccoglierle in ordine chiaro e leggibile.

È facile e rapido scrivere poche linee di programma che funzionino subito, immediatamente comprensibili da qualcun altro. In effetti, se un programma è molto breve, non è importante dedicare grande cura alla strutturazione. Il problema sorge con i programmi lunghi e impegnativi. In questo caso, scrivendo un programma "di getto", è estremamente facile perdersi, a meno che non si abbia una memoria formidabile.

Ciò che serve è un po' di calma: sedersi e progettare sistematicamente il programma. Molti programmatori partono con un'idea vaga di ciò che si vuole dal computer, e, quanto più la richiesta è complessa, tanto più vaga è l'idea.

Il concetto di base, sia per un gioco che per programma commerciale, è che la quantità di materiale è troppo vasta per essere tenuta a mente tutta in una volta.

Mettiamoci lontani dal computer (o meglio, per evitare tentazioni, in un'altra stanza) e buttiamo giù in modo generico quello che vogliamo fare. Questo è il fondamento della cosiddetta analisi del progetto. Per esempio potremmo scrivere:

Sistema indicizzato
Il programma deve permettere di creare, aggiornare, cancellare, riordinare ed elencare i record in memoria. L'accesso ai record dovrebbe avvenire tramite una parola chiave. Si deve essere in grado di scrivere e rileggere il complesso dei record usando un file.

Adesso occorre scomporre questa descrizione generica in pochi "passi logici" o moduli. Le operazioni raccolte in ogni modulo saranno probabilmente ancora complesse e andranno così scomposte a loro volta in sezioni più piccole, finché ogni livello più basso sia abbastanza semplice da codificare. La **figura 1** mostra come si potrebbe far questo con l'indice sistematico.

Ognuna delle sezioni più piccole non dovrebbe occupare più di una pagina di scritto (circa 60 linee), ma la metà è ancor meglio: ognuno di questi moduli a basso livello deve essere di facile comprensione. Alla fine, ogni modulo costituirà una subroutine nel programma.

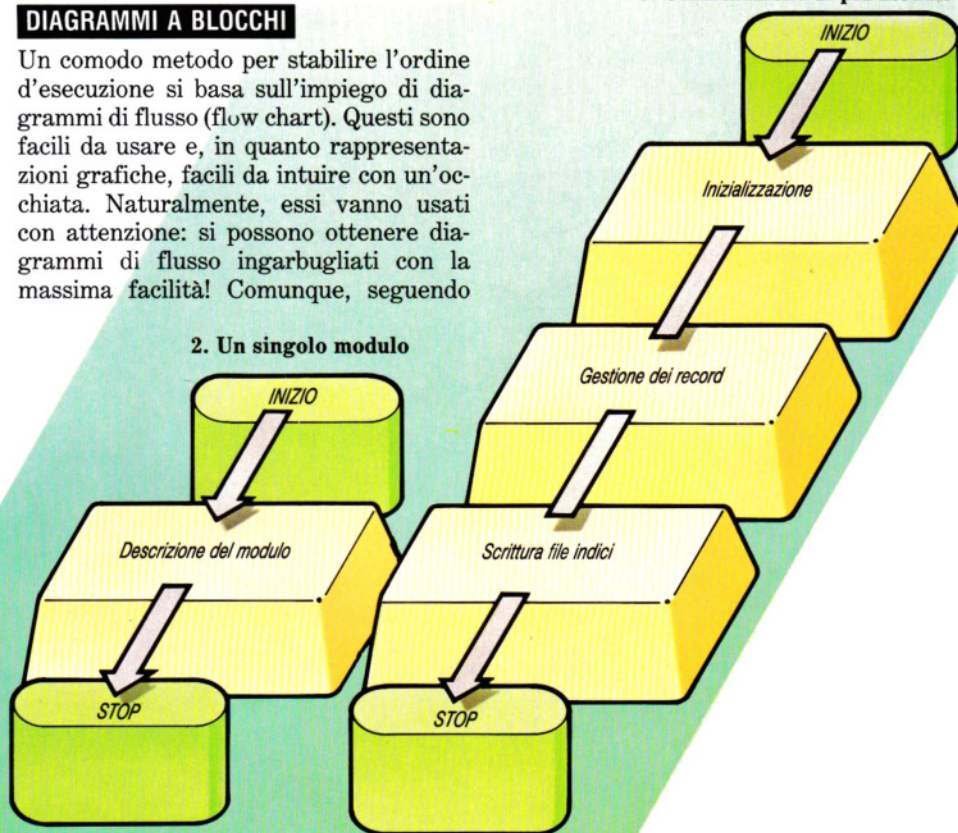
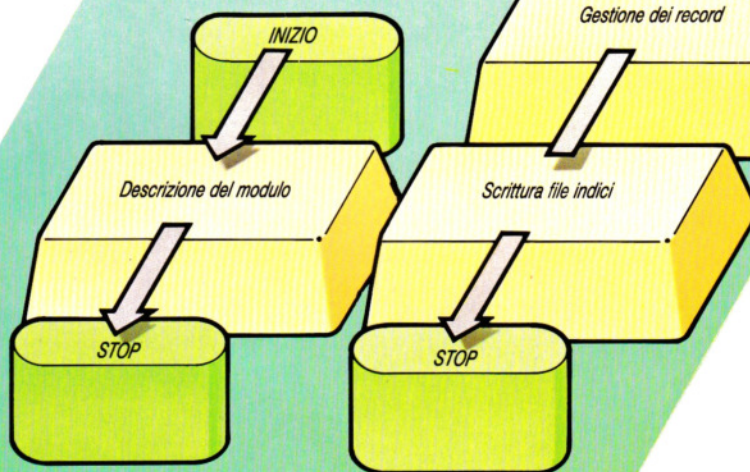
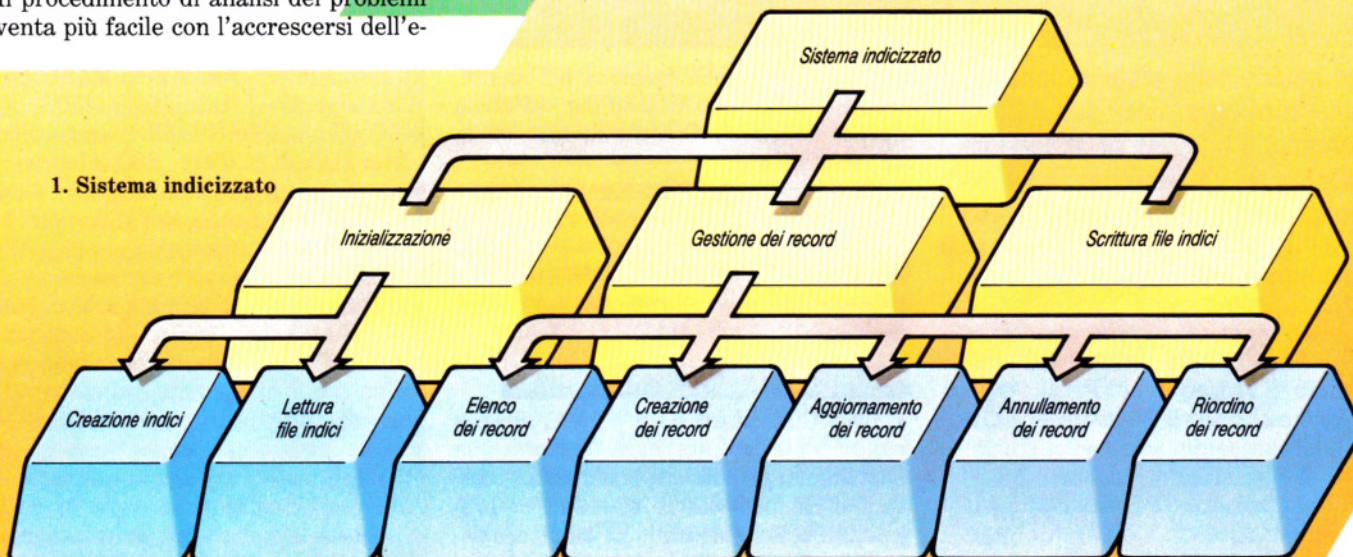
Il procedimento di analisi dei problemi diventa più facile con l'accrescersi dell'esperienza e, come sempre, il miglior modo per apprenderlo è quello di applicarlo. Questo metodo di scomposizione è noto come progettazione 'dall'alto al basso': siamo partiti dall'alto (la descrizione generica del programma) e ci siamo mossi verso il basso (i livelli più bassi del programma). Non abbiamo ancora deciso l'ordine dei moduli, cioè l'ordine in cui verranno eseguiti dal programma: questo è il passo successivo.

DIAGRAMMI A BLOCCHI

Un comodo metodo per stabilire l'ordine d'esecuzione si basa sull'impiego di diagrammi di flusso (flow chart). Questi sono facili da usare e, in quanto rappresentazioni grafiche, facili da intuire con un'occhiata. Naturalmente, essi vanno usati con attenzione: si possono ottenere diagrammi di flusso ingarbugliati con la massima facilità! Comunque, seguendo alcune piccole regole, si può ordinare un programma in modo chiaro e strutturato.

La **figura 2** mostra come si può specificare un singolo modulo. Il programma segue le linee in direzione delle frecce e le 'scatole' descrivono quello che succede a ogni livello. Per eseguire una serie di moduli in sequenza basta aggiungere più blocchi nel corretto ordine tra l'inizio e la fine, **figura 3**.

2. Un singolo modulo

3. Combinazione di più moduli

1. Sistema indicizzato

Un diagramma a blocchi semplice come questo va bene per un programma in cui non vanno prese decisioni, ma la capacità di un computer risiede proprio nella sua abilità di fare scelte in un programma. Ecco quindi la familiare istruzione IF ... THEN, che è la più usata delle strutture in BASIC.

Osserviamo ora tutte le diverse strutture e il modo di rappresentarle con diagrammi a blocchi.

IF ... THEN ... ELSE

Sebbene vi siano differenze nel modo in cui i vari computer usano questo comando, IF ... THEN ... ELSE è la base di tutte le decisioni che il computer deve prendere. La **figura 4** mostra il suo diagramma di flusso, che in BASIC si scrive:

```
100 IF condizione THEN istruzioni1 ELSE
    istruzioni2
```

Ciò significa che se la condizione è vera, allora si esegue l'istruzione 1, altrimenti l'istruzione 2.

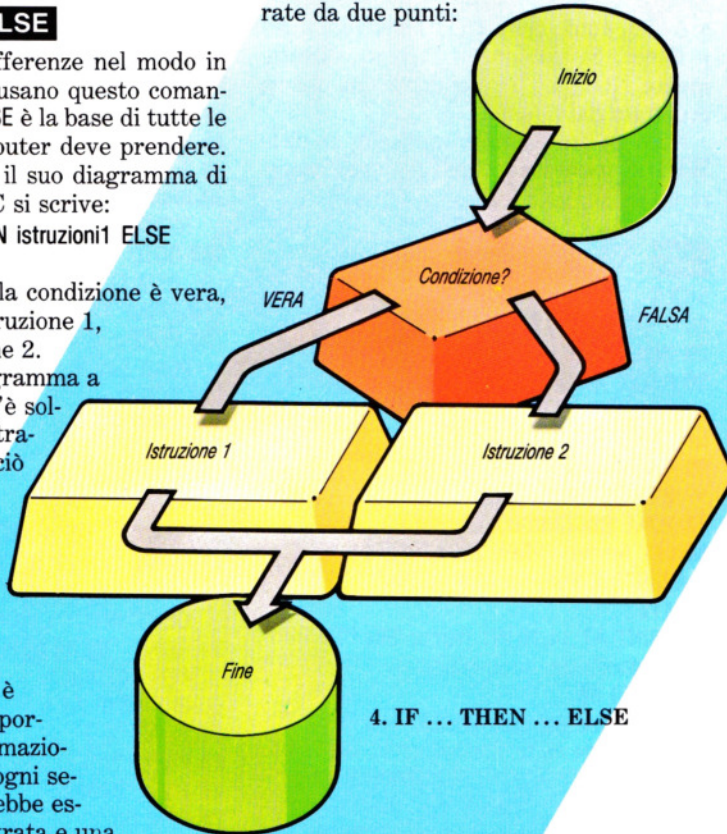
Osservando il diagramma a blocchi si nota che c'è soltanto un punto d'entrata e uno di uscita e ciò semplifica molto la riprova e la correzione visto che si sa esattamente dove questa sezione di programma deve iniziare a finire. Questa in effetti è una regola molto importante della programmazione strutturata: per ogni sezione di codice dovrebbe esserci soltanto un'entrata e una uscita.

La maggior parte delle versioni di BASIC non possiede l'ELSE, non ce l'ha lo Spectrum, né lo ZX81, né il Commodore 64 o il Vic; si può usare, però, una GOTO:

```
100 ...
110 IF condizione THEN GOTO 140
120 istruzioni2 :REM questa parte rappresenta
    la ELSE
130 GOTO 150
140 istruzioni1 :REM questa parte rappresenta
    la THEN
150 ...
```

Inoltre, più di una istruzione può essere inclusa nella parte di THEN e di ELSE. Per esempio, ecco una sezione di programma per ordinare correttamente due numeri: è la base di una routine di ordinamento alfabetico che sarà presentata nella seconda parte di questa lezione. In questo caso, la parte ELSE ha quattro istruzioni:

```
100 IF primo< =secondo THEN GOTO 160
110 LET temporaneao=primo
120 LET primo=secondo
130 LET secondo=temporaneo
140 LET ordine$="errato"
150 GOTO 170
160 LET ordine$="giusto"
170 ...
```

Si potrebbe anche usare ELSE, ma tutto andrebbe scritto su una sola linea. Sono permesse istruzioni multiple purché separate da due punti:

4. IF ... THEN ... ELSE

```
100 IF primo>secondo THEN temporaneo=
    primo: primo=secondo: secondo=
    temporaneo: ordine$=
    "errato" ELSE ordine$="giusto"
```

Come si vede, non è facile leggere e capire programmi scritti usando istruzioni così complesse: finché è possibile andrebbero evitate.

Infine, in molti casi può non servire del tutto la parte ELSE. Il diagramma a blocchi della **figura 5** risulta:

```
100 IF condizione THEN istruzioni
```

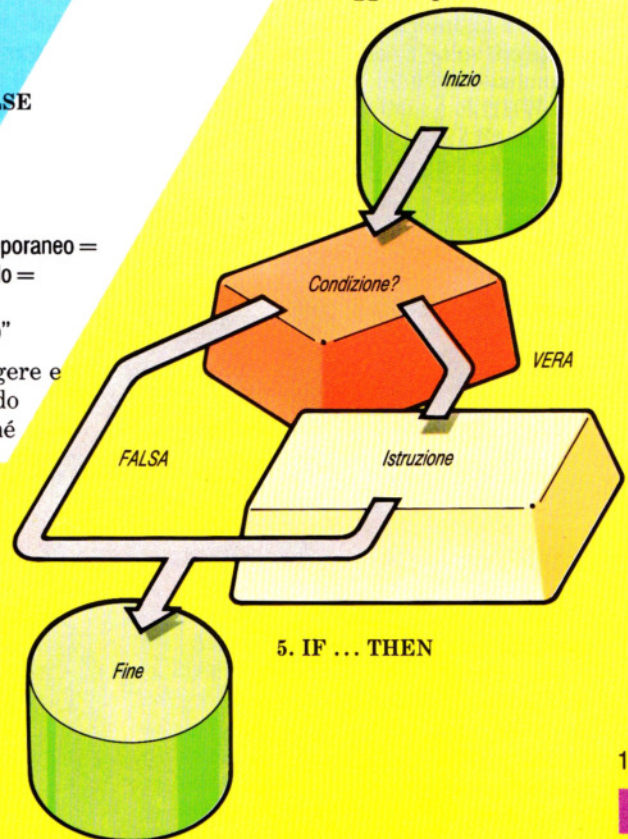
che corrisponde alla semplice struttura: IF ... THEN.

STRUTTURE NIDIFICATE

Le linee IF ... THEN ... ELSE possono essere 'nidificate', ossia una o ambedue le istruzioni tra cui la linea IF ... THEN sceglie possono essere esse stesse delle IF ... THEN. Per esempio, questa sezione di programmi registra le vittorie riportate da due giocatori e, dopo ogni partita, stampa i risultati:

```
100 IF T1< >T2 THEN GOTO 130
110 PRINT "Pareggio!"
120 GOTO 190
130 IF T1<T2 THEN GOTO 170
140 PRINT "Ha vinto il primo giocatore"
150 LET P1=P1+1
160 GOTO 190
170 PRINT "Ha vinto il secondo giocatore"
180 LET P2=P2+1
190 ...
```

Tutte le strutture si possono nidificare in qualsiasi combinazione e, in teoria, ad ogni livello. Tuttavia, più nidificazioni si fanno, meno leggibile diventa il programma, per cui conviene limitarsi a un livello di tre o quattro strutture. Se ne sono necessarie di più, è preferibile suddividere il programma in moduli o subroutine di minor grandezza.

Consideriamo ancora l'ultimo programma. È in effetti difficile seguirne lo svolgimento anche se è strutturato perfettamente. Un modo leggibile per includere le

5. IF ... THEN

istruzioni una nell'altra è far rientrare le linee di programma. Ciò è possibile soltanto sugli Acorn e sullo Spectrum, in cui si può riscrivere il programma così:

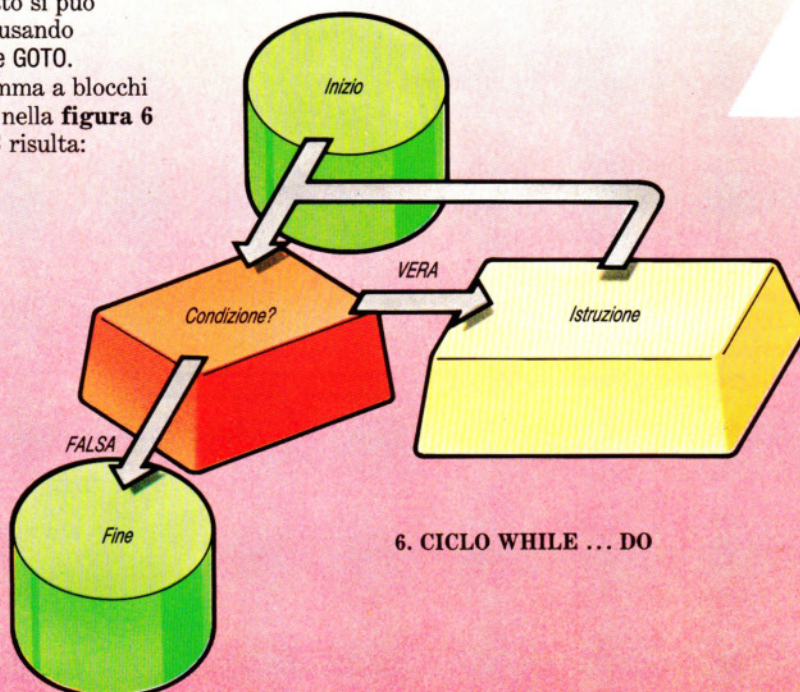
```
100 IF T1< >T2 THEN GOTO 130
110 PRINT "Pareggio!"
120 GOTO 190
130 IF T1<T2 THEN GOTO 170
140 PRINT "Ha vinto il primo giocatore"
150 LET P1=P1+1
160 GOTO 190
170 PRINT "Ha vinto il secondo giocatore"
180 LET P2=P2+1
190 ...
```

Sui computer Acorn si possono evidenziare le istruzioni strutturate scrivendo LISTO 7 seguito da LIST.

Altri nomi per rendere più appariscenti le strutture di un programma sono l'inserimento di linee vuote intorno a sezioni diverse e l'uso di frasi REM. Le linee vuote si inseriscono, con gli Acorn o lo Spectrum, digitando il numero di linea seguito da uno spazio e da RETURN o ENTER. Un effetto simile si ottiene sul Dragon, sul Tandy e sui Commodore digitando due punti (:) anziché lo spazio.

WHILE ... DO

Un'altra struttura essenziale è WHILE ... DO, che serve a creare nei programmi cicli iterativi in uno dei modi più utili. Le istruzoni all'interno dei ciclo vengono ripetute più volte, finché (WHILE) una certa condizione è vera. Purtroppo la struttura WHILE ... DO non esiste nella gran parte dei BASIC, ma lo stesso effetto si può riprodurre usando IF ... THEN e GOTO.

Il diagramma a blocchi è mostrato nella **figura 6** e in BASIC risulta:

```
100 ...
110 IF NOT(condizione) THEN GOTO 140
120 istruzioni
130 GOTO 110
140 ...
```

Si noti che la linea 110 contiene IF NOT (condizione) ossia viene fatto un controllo su quando una particolare condizione *non* è vera, contrariamente al solito. Ciò non costituisce un reale problema: se la condizione è A = B, allora NOT (A = B) equivale a A⟨⟩B, come NOT (A⟩B) equivale a A⟩ = B ecc. Si può anche scrivere direttamente NOT(A = B), poiché è una forma accettata dal BASIC.

Ecco un breve programma con un ciclo WHILE da usarsi come timer per cuocere un uovo:

```
5 CLS
10 PRINT AT 3,11;"CONTATEMPO"
20 INPUT "Quanti minuti devo contare?",t
30 PRINT AT 7,5;"Premere qualsiasi tasto"
40 PAUSE 0
50 CLS
60 PRINT FLASH 1;AT 10,9;"□
   STO CONTANDO□"
70 POKE 23672,0: POKE 23673,0
80 LET tempo=PEEK 23672+256* PEEK
   23673:IF tempo>t*50*60 THEN GOTO
   110
90 PRINT AT 14,10;INT (tempo/50); "□
   secondi"
100 GOTO 80
105 REM fine del ciclo WHILE
110 PRINT FLASH 1;AT 14, 10;"□ECCO
   FATTO□"
120 BEEP .5,20
```

```
5 PRINT "♡■":POKE 53281,4
10 PRINT TAB(16)"▣CONTATEMPO"
20 INPUT "▣▣▣□□□□□
   QUANTI MINUTI DEVO CONTARE";T
30 PRINT TAB(8)"▣▣▣PREMERE UNO
   SPAZIO PER INIZIARE"
40 GET A$:IF A$=" " THEN GOTO 40
50 PRINT "♡"
60 PRINT TAB(15)"STO CONTANDO"
70 TI$="000000"
75 REM INIZIO DEL CICLO WHILE
80 IF VAL(TI$)=>T*100 THEN GOTO 110
90 PRINT "▤▣▣▣▣▣"TAB(15);
   RIGHT$(TI$,2)"□SECONDI"
100 GOTO 80
105 REM FINE DEL CICLO WHILE
110 PRINT TAB(16)"▣▣▣ECCO FATTO!"
120 POKE 54296,15:POKE 54278,128:
   POKE 54276,17:POKE 54273,50
130 FOR D=1 TO 200: NEXT:POKE 54276,0:
   POKE 54278,0
```

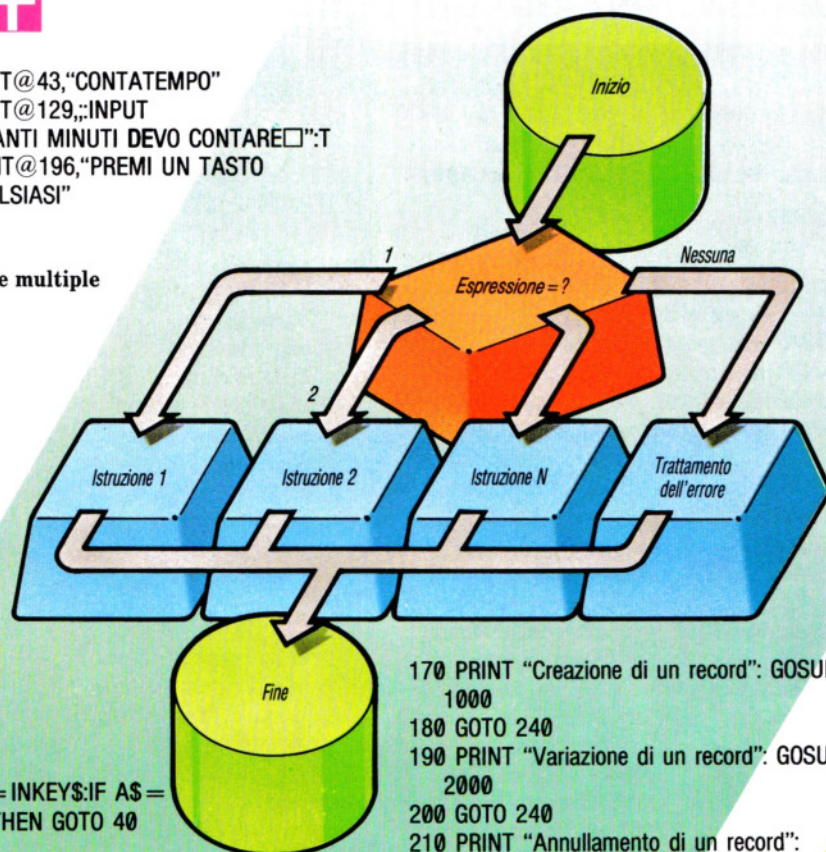
```
5 PRINT "♡■":POKE 36879,29
10 PRINT TAB(7)"▣CONTATEMPO"
20 PRINT "▣▣▣QUANTI MINUTI
   DEVO":INPUT"□CONTARE";T
30 PRINT "▣▣▣PREMERE UNO SPAZIO
   PER INIZIARE"
40 GET A$:IF A$=" " THEN 40
50 PRINT "♡"
60 PRINT TAB(8)"STO CONTANDO"
70 TI$="000000"
75 REM INIZIO DEL CICLO WHILE
80 IF VAL(TI$)=>T*100 THEN 110
90 PRINT "▤▣▣▣▣▣"TAB(6);
   RIGHT$(TI$,2)"□SECONDI"
100 GOTO 80
105 REM FINE DEL CICLO WHILE
110 PRINT TAB(6)"▣▣▣ECCO FATTO!"
120 POKE36878,15:POKE36876,200
130 FOR D=1 TO 200:NEXT:POKE36876,0
```

```
5 CLS
10 PRINT TAB(15,2) "CONTATEMPO"
20 INPUT TAB(5,4) "QUANTI MINUTI DEVO
   CONTARE□",T
30 PRINT TAB(7,6) "PREMERE UNO SPAZIO
   PER INIZIARE"
40 K$=GET$
50 CLS:VDU23;8202;0;0;0;
60 PRINT TAB(15,10) "STO CONTANDO"
70 TIME=0
75 REM inizio del ciclo WHILE
```

6. CICLO WHILE ... DO

```
80 IF TIME>T*60*100 THEN GOTO 110
90 PRINT TAB(13,12);INT(TIME/100);"□
   secondi"
100 GOTO 80
105 REM fine del ciclo WHILE
110 PRINT TAB(14,15) "ECCO FATTO!"
120 VDU7
```

```
5 CLS
10 PRINT@43,"CONTATEMPO"
20 PRINT@129,;:INPUT
   "QUANTI MINUTI DEVO CONTARE□":T
30 PRINT@196,"PREMI UN TASTO
   QUALSIASI"
```

7. Scelte multiple

```
40 A$=INKEY$:IF A$=
   "" THEN GOTO 40
50 CLS
60 PRINT@235,"STO CONTANDO"
70 TIMER=0
75 REM INIZIO DEL CICLO WHILE
80 IF TIMER>T*3000 THEN GOTO 110
90 PRINT@298,INT(TIMER/50);"SECONDI"
100 GOTO 80
105 REM FINE DEL CICLO WHILE
110 PRINT@364,"ECCO FATTO!"
120 SOUND 180,3
```

SCELTE MULTIPLE

Le strutture IF ... THEN e WHILE ... DO sono, in genere, sufficienti per quasi tutti i programmi. Ci sono però alcune altre strutture che semplificano maggiormente la programmazione. Per esempio, spesso ci sono più di due percorsi possibili a partire da un punto particolare di un programma. Si potrebbe sfruttare l'uso di IF ... THEN nidificati, ma è più comodo usare una struttura a scelte multiple.

Il diagramma a blocchi di questa struttura, chiamata CASE, è mostrato nella **figura 7** e in BASIC si può scrivere così:

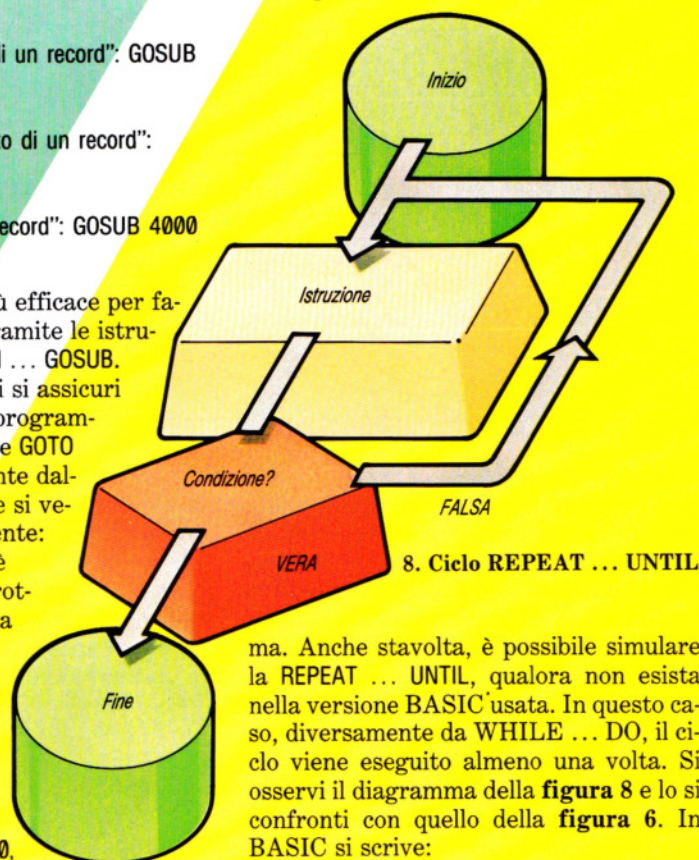
```
100 REM blocco delle scelte
110 IF C$="C" THEN GOTO 170
120 IF C$="V" THEN GOTO 190
130 IF C$="A" THEN GOTO 210
140 IF C$="E" THEN GOTO 230
150 PRINT "Comando non riconosciuto"
160 GOTO 240
170 PRINT "Creazione di un record": GOSUB
    1000
180 GOTO 240
190 PRINT "Variazione di un record": GOSUB
    2000
200 GOTO 240
210 PRINT "Annullamento di un record":
    GOSUB 3000
220 GOTO 240
230 PRINT "Elenco dei record": GOSUB 4000
240 REM fine del blocco
```

Un metodo ancora più efficace per fare scelte multiple è tramite le istruzioni ON ... GOTO e ON ... GOSUB. Usando ON ... GOTO ci si assicuri che ogni opzione del programma contenga adeguate GOTO per uscire regolarmente dall'opzione stessa, come si vede nell'esempio seguente: dopo ogni opzione, c'è una GOTO 1210 per dirottare il programma alla fine della routine.

```
1000 REM SUBROUTINE
     PER UN POLIGONO
1010 INPUT "QUANTI
     LATI VUOI";N
1020 ON N-2 GOTO
     1060,1100,1140,1160,
     1180,1200
1030 PRINT "NON CONOSCO IL NOME DI
     UN"
1040 PRINT "POLIGONO CON□";N;"□LATI."
1050 GOTO 1210
1060 PRINT "È UN TRIANGOLO"
1070 PRINT "UN TRIANGOLO CON LATI
     UGUALI"
1080 PRINT "SI CHIAMA TRIANGOLO
     EQUILATERO."
1090 GOTO 1210
1100 PRINT "È UN QUADRILATERO"
1110 PRINT "UN QUADRILATERO CON LATI
     ED ANGOLI UGUALI"
1120 PRINT "SI CHIAMA QUADRATO"
1130 GOTO 1210
1140 PRINT "È UN PENTAGONO"
1150 GOTO 1210
1160 PRINT "È UN ESAGONO"
1170 GOTO 1210
1180 PRINT "È UN ETTAGONO"
1190 GOTO 1210
1200 PRINT "È UN OTTAGONO"
1210 PRINT
1220 RETURN
```

Poiché si tratta di una subroutine, occorre un programma "chiamante", che compare nel successivo paragrafo.

REPEAT ... UNTIL

La struttura REPEAT ... UNTIL è un altro modo per creare un ciclo in un program-

8. Ciclo REPEAT ... UNTIL

ma. Anche stavolta, è possibile simulare la REPEAT ... UNTIL, qualora non esista nella versione BASIC usata. In questo caso, diversamente da WHILE ... DO, il ciclo viene eseguito almeno una volta. Si osservi il diagramma della **figura 8** e lo si confronti con quello della **figura 6**. In BASIC si scrive:

```
100...
110 istruzioni
120 IF NOT(condizione) THEN GOTO 110
130...
```

Usando la subroutine dell'ultimo esempio, ecco come si può scrivere un programma con un ciclo REPEAT:

```
10 PRINT "POSSO DIRTI I NOMI"
20 PRINT "DI ALCUNI POLIGONI."
30 REM INIZIO DEL CICLO
40 GOSUB 1000
50 INPUT "VUOI UN ALTRO NOME□";A$
60 IF LEFT$(A$,1)="S" THEN GOTO 30
70 PRINT "CIAO!":END
```

La linea 1000 è la subroutine dei poligoni di poco fa.

Si noti che su alcuni computer BBC può essere necessario cambiare il punto e virgola in virgola alla linea 50 del programma e alla 1010 della subroutine. Il BASIC BBC ha l'istruzione REPEAT ... UNTIL e per di più la parte UNTIL dell'istruzione può non trovarsi sulla stessa linea di REPEAT, diversamente dall'istruzione IF ... THEN ... ELSE.

Perciò, l'ultimo esempio potrebbe essere scritto così:

```
10 PRINT "Posso dirti i nomi"
20 PRINT "di alcuni poligoni."
30 REPEAT
40 GOSUB 1000
50 INPUT "Vuoi un altro nome□",A$
60 UNTIL LEFT$(A$,1)<>"S"
70 PRINT "Ciao!"
```

Così è ancora più facile da comprendere.

CICLI FOR ... NEXT

Il familiare ciclo FOR ... NEXT, in realtà, non è che un caso particolare nel ciclo WHILE ... DO: lo si usa, infatti, quando si conosce in anticipo il numero di passaggi attraverso il ciclo, che va quindi specificato all'inizio. La variabile che tiene conto dei passaggi è detta *variabile di controllo*.

Un diagramma a blocchi di un ciclo FOR ... NEXT può presentarsi come nella **figura 9**. Confrontandolo con il ciclo WHILE della **figura 6**, si vede che ha la stessa struttura globale. In BASIC si scrive:

```
100 FOR i=min TO max STEP val
110 istruzioni
120 NEXT i
```

Uscire da un ciclo FOR ... NEXT con una GOTO è una pessima abitudine, poiché il BASIC non ha modo di accorgersi di ciò che abbiamo fatto. Peggio ancora è rientrare a metà del ciclo, dopo esserne usciti: un simile programma sarebbe di difficile comprensione. Si evitino simili passaggi pericolosi!

RIUNIAMO TUTTO ASSIEME

Le strutture presentate sono sufficienti, per qualsiasi programma si stia scrivendo. Ma il cammino è ancora lungo prima di completare un programma. Devono essere specificate le variabili che si intendono usare ed occorre verificare che le variabili usate in moduli diversi non siano in contraddizione tra loro. Poi è necessario stabilire quale sarà l'input e l'output del programma. Infine, va provato ogni singolo modulo, prima di collegarlo ad altri. Come fare ciò mantenendo un programma leggibile e strutturato sarà detto nella seconda parte della lezione. Intanto, si dia una attenta scorsa a qualche programma per riconoscerne le strutture. Si può provare anche a migliorarne uno rendendolo più strutturato, scegliendolo tra i propri o tra quelli delle riviste.

D+R

La forma dei simboli usati nei diagrammi di flusso ha un significato speciale?

Sì, i diagrammi a blocchi seguono una convenzione. Le forme principali sono cinque:
1. I rettangoli arrotondati sono detti *terminali* e indicano l'inizio e la fine di un programma.
2. I cerchi sono *connettori*, usati all'inizio e alla fine di un modulo.
3. I rettangoli rappresentano le operazioni e contengono le *istruzioni* del programma.
4. I rombi indicano *decisioni*: da essi escono almeno due percorsi, secondo la decisione presa nel blocco.
5. Infine i parallelogrammi (che in questa lezione non compaiono) rappresentano operazioni di *input/output*, cioè ogni informazione immessa nel programma e ogni uscita su schermo o su stampante.

9. Ciclo FOR ... NEXT

NUMERI SOTTO ZERO!

- QUANDO I NUMERI NEGATIVI SONO NECESSARI
- PROGRAMMA DI CONVERSIONE PER NUMERI NEGATIVI
- LA CONVENZIONE SUL SEGNO

I numeri binari ed esadecimali non sono difficili da comprendere, ma esprimere valori negativi in questi sistemi di numerazione può causare qualche problema.

Nella programmazione dei giochi possono essere necessari valori negativi, per esempio per spostarsi sullo schermo in determinate direzioni. Queste vanno codificate in byte di otto bit, in quanto il computer non ha altri modi per memorizzare dati in memoria. Ma sorge un problema: come si è visto, un byte può rappresentare tutti i numeri da Ø a 255, ossia ØØØØØØØØ e 11111111. Con questo però si esauriscono le possibiiltà del binario a otto bit, che non ha spazio per un segno "meno" o "più", né modo di rappresentarlo.

Nell'aritmetica comune, sottraendo 1 da Ø si ottiene −1; ecco lo stesso calcolo in binario di otto bit:

```
ØØØØØØØØ
      −1
--------
11111111
```

Si può provare a chiedere uno in prestito dalla colonna più a sinistra, ma quando il numero binario è limitato a otto bit non esiste modo di far niente. Similmente, sottraendo un altro 1 (in aritmetica tradizionale si otterrebbe −2) abbiamo 1111111Ø, che in binario corrisponde al decimale 254 dato che 11111111 è 255.

Non sono stranezze tipiche del binario: per esempio, si immagini cosa accadrebbe in decimale se si avessero solo tre posti, o 'colonne' in cui mettere i numeri. Ecco cosa succederebbe se si provasse ad aggiungere 999 a 1ØØ con queste limitazioni:

```
  1ØØ
 +999
------
(1)Ø99
```

L'uno nella colonna delle migliaia è scritto tra parentesi: in un sistema aritmetico con tre sole colonne non c'è spazio per il riporto. Il risultato di questa addizione, in un sistema a tre posti, è quindi 99, ossia esattamente il risultato di 1ØØ meno 1!

Allo stesso modo, sommare 998 a 1ØØ in un sistema a tre posti corrisponde a sottrarre 2 e sottrarre 998 da 1ØØ corrisponde a sommare 2.

In conclusione, la stessa fila di numeri in un byte di otto bit può rappresentare un numero negativo o positivo, nonostante la confusione e la difficoltà nel calcolo.

Come fare allora? Nella maggior parte delle applicazioni degli home computer il problema non si pone: indirizzi di memoria e codici di operazione, ambedue resi in binario, si possono sempre considerare positivi. Le sole volte in cui si incontrano numeri negativi sono nelle DATA o nei salti, equivalenti in codice macchina elle GOTO del BASIC.

RIGIRARE I BIT

Il procedimento in binario per passare dal valore di un numero positivo al suo negativo si chiama *complemento 2*: non ha basi teoriche, perlomeno non di facile comprensione, ma funziona.

Per passare da un numero binario al suo valore negativo, si "girano" i bit e si aggiunge 1. "Girare" un bit significa cambiare il suo valore da Ø a 1 e viceversa.

Il seguente programma ha proprio questo scopo. Si noti che, quando il binario è limitato a otto cifre, il suo equivalente hex è esattamente di due cifre: ciò vuol dire che anche l'equivalente hex del comportamento 2 corrisponde al negativo.

```
10 PRINT AT 0,7;"NUMERI NEGATIVI"
20 PRINT AT 2,1;"DEC";TAB 14;"BIN";
   TAB 28;"HEX"
30 LET A$="□□□□□□□□□□□□□
   □□□□□□"
40 FOR N=7 TO 13
50 PRINT AT N,6; INVERSE 1;A$
60 NEXT N
70 PRINT AT 8,8;"COMPLEMENTO"
80 PRINT AT 10,3;"+";AT 10,30;"+"
90 PRINT AT 15,8;"COMPLEMENTO A 2"
95 LET C=Ø
100 LET DD=-C: DIM A(8)
110 PRINT AT 4,Ø;"□□□□";AT 4,4-LEN
   STR$ C;C
115 POKE 23608,C: LET E=PEEK 23608:
   LET Z=E: GOSUB 300: PRINT AT 4,29;A$
120 PRINT AT 17,Ø;"□□□□";AT 17,4
   -LEN STR$ DD;DD
130 LET D=128: LET CC=E
140 FOR N=1 TO 8: LET A(N)=Ø
150 IF CC-D>=Ø THEN LET A(N)=1:
   LET CC=CC-D
160 PRINT BRIGHT 1;AT 4,6+2*N;A(N);
   AT 10,6+2*N;1-A(N)
170 LET D=D/2: NEXT N
180 POKE 23608,DD: LET DD=PEEK 23608:
   LET D=128
185 LET Z=DD: GOSUB 300: PRINT AT
   17,29;A$
190 FOR N=1 TO 8: LET B=Ø: IF DD-D
   >=Ø THEN LET B=1: LET DD=DD
   -D
200 PRINT BRIGHT 1;AT 17,6+2*N;B: LET D
   =D/2: NEXT N
210 PRINT AT 18,Ø;"----□□---
   -----------------
   -□□----"
220 PRINT AT 19,3;"Ø□□□□Ø□Ø□Ø□Ø□
   Ø□Ø□Ø□Ø□□□□□□□ØØ"
230 IF INKEY$="" THEN GOTO 230
240 LET A$=INKEY$: IF A$="□" THEN
   LET C=C+1: IF C=128 THEN LET C=
   -128: BEEP 1,1
250 IF A$="B" OR A$="b" THEN LET C
   =C-1: IF C=-129 THEN LET C
   =127: BEEP 1,1
260 IF A$<>"□" AND A$<>"B" AND
   A$<>"b" THEN INPUT "?";C
270 GOTO 100
300 LET ZA=INT (Z/16): LET ZB=Z-
   (16*ZA)
310 LET ZA+48: IF ZA>57 THEN LET ZA
   =ZA+7
320 LET ZB=ZB+48: IF ZB>57 THEN LET
   ZB=ZB+7
330 LET A$=CHR$ ZA: LET A$=A$
   +CHR$ ZB: RETURN
```

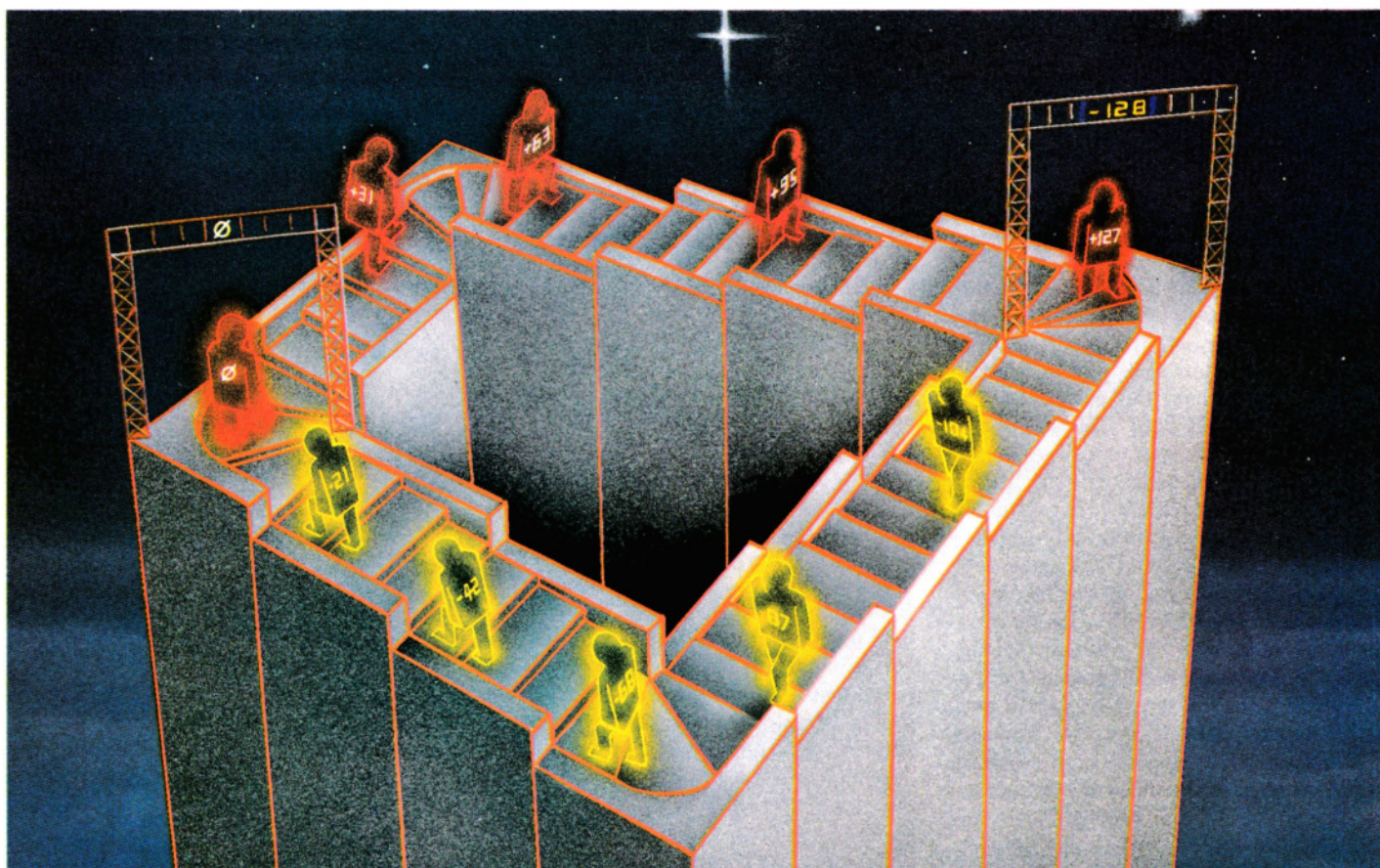
```
10 PRINT AT 0,7;"NUMERI NEGATIVI"
20 PRINT AT 21,1;"DEC"; TAB 14;"BIN";TAB
   28;"HEX"
30 LET A$="■■■■■■■■■■
   ■■■■■■■■■■■"
40 FOR N=7 TO 13
50 PRINT AT N,6; A$
60 NEXT N
70 PRINT AT 8,8;"COMPLEMENTO□";AT
   12,21;"+1"
80 PRINT AT 10,3;"+"; AT 10,30;"+"
90 PRINT AT 15,8;"COMPLEMENTO A2"
95 LET C=Ø
100 LET DD=-C
105 DIM A(8)
110 PRINT AT 4,Ø;"□□□□"; AT 4,4-LEN
   STR$ C;C
115 POKE 16507, C
116 LET E=PEEK 16507
117 LET Z=E
118 GOSUB 300
119 PRINT AT 4,29;A$
120 PRINT AT 17,Ø;"□□□□"; AT 17,4
   -LEN STR$ DD;DD
130 LET D=128
135 LET CC=E
140 FOR N=1 TO 8
145 LET A(N)=Ø
150 IF CC-D>=Ø THEN LET A(N)=1
155 IF CC-D>=Ø THEN LET CC=CC-D
160 PRINT AT 4,6+2*N;A(N);AT 10,6+
   2*N; 1-A(N)
170 LET D=D/2
175 NEXT N
```

```
180 POKE 16507,DD
181 LET DD=PEEK 16507
182 LET D=128
185 LET Z=DD
186 GOSUB 300
187 PRINT AT 17,29;A$
190 FOR N=1 TO 8
191 LET B=0
192 IF DD−D>=0 THEN LET B=1
193 IF DD−D>=0 THEN LET DD=
    DD−D
200 PRINT AT 17,6+2*N;B
205 LET D=D/2
207 NEXT N
210 PRINT AT 18,0;"− − − − □□ − − −
    − − − − − − − − − − − − − − − −
    − □□ − − − −"
220 PRINT AT 19,3;"0□□□□0□0□0□0□
    0□0□0□0□□□□□□□00"
230 IF INKEY$="" THEN GOTO 230
240 LET A$=INKEY$
242 IF A$="F" THEN LET C=C+1
244 IF A$="F" AND C=128 THEN LET C
    =128
250 IF A$="B" THEN LET C=C−1
255 IF A$="B" AND C=−129 THEN LET
    C=127
260 IF A$<>"F" AND A$<>"B" THEN
    INPUT C
270 GOTO 100
300 LET ZA=INT(Z/16)
305 LET ZB=Z−(16*ZA)
310 LET ZA=ZA+28
320 LET ZB=ZB+28
330 LET A$=CHR$ ZA
340 LET A$=A$+CHR$ ZB
350 RETURN
```

```
20 POKE54277,33:POKE54278,255:POKE
   54273+23,15:POKE276,33:POKE54273,0
30 FOR Z=1 TO 8:READ A(Z):NEXT Z:
   DATA 128,64,32,16,8,4,1
40 Z$="0123456789ABCDEF":A=0:A$="
   +−":AA=1:POKE650,128:V$="▮▮
   ▮▮▮▮▮▮▮"
50 POKE 53280,1:POKE 53281,1:PRINT "♡
   ■▣▣▣▣"
60 FOR Z=1 TO 9:PRINT TAB(5)"▣□□□
   □□□□□□□□□□□□□□□□□□□
   □□□□□□□□":NEXT Z
70 PRINT "▣▣" TAB(13)"COMPLEMENTO
   A 2"
80 PRINT "▣" TAB(12)"NUMERI NEGATIVI"
90 PRINT TAB(12)"− − − − − − − − −
   − − − −"
100 PRINT "▩□□DEC□□□□□□□□□□
    □□□□BIN□□□□□□□□□□□□
    □□ESA"
110 PRINT "▣▣▣▣▣▮▮▮▮▮
    ▮▮▮COMPLEMENTO":PRINT"▣▣
    ▣"TAB(29)"+1"
120 PRINT"○○○○▮▮+"TAB(37)"
    +"
130 PRINT"▣▣▣▣▣▣▣▣▣▣−
    − − − − − − − − − − − − − − − −
    − − − − − − − − − − − − − − − −
    − − − − − − − − − − −";
140 PRINT"□□0□□□□□□□0□□□0□□0
    □□0□□0□□0□□0□□0□□□□□
    □00":GOTO300
150 GET K$:IF K$=""THEN 150
160 S=0:SS=0:IF K$<>"□" AND K$<
    >"B" THEN 500
170 A1=A:IF K$<>"□" THEN 210
180 IF AA=2 THEN A1=A1−1
190 IF AA=1 THEN A1=A1+1
200 GOTO 240
210 IF K$<>"B" THEN 240
220 IF AA=2 THEN A1=A1+1
230 IF AA=1 THEN A1=A1−1
240 IF A1<0 OR A1>128 THEN AA=AA
    +1
250 IF A1>128 THEN A1=127
260 IF A1<0 THEN A1=1
270 A=A1
280 IF A=0 THEN AA=1
290 IF A=128 THEN AA=2:POKE54273,9:
    FORZ=1TO20:NEXT:POKE54273,0
300 IF AA>2 THEN AA=1
310 W=0:IF AA=1 THEN T=1:TT=0
320 IF AA=2 THEN T=0:TT=1:W=1
330 A1=A−W:FOR Z=1 TO 8:IFA(Z)<=
    A1 THEN B(Z)=T:C(Z)=TT:A1=A1−A
    (Z):GOTO350
340 B(Z)=TT:C(Z)=T
350 IF B(Z)=1THENS=S+A(Z)
360 NEXT Z:A2=0:FORZ=1TO8:IFC(Z)=
    0 THEN A2=A2+A(Z)
370 NEXT Z:A1=(255−A2)+T:IFA1>
    255 THENA1=0
380 FORZ=1TO 8:IFA(Z)<=A1+
    W THEN D(Z)=1:A1=A1−A(Z):SS=SS
    +A(Z):GOTO400
390 D(Z)=0
400 NEXT Z
410 PRINT"▣▣▣▣▣";MID$(A$,AA,1)
    "□□□□▮▮▮▮▮▮▮"A
420 PRINT"○"V$;:FORZ=1TO8:PRINTB(Z);:
    NEXT Z
430 ZA=INT(S/16):ZB=S−(16*ZA):PRINT
    "□□□□□"MID$(Z$,ZA+1,1);MID$(Z$,
    ZB+1,1)
440 PRINT"▣▣▣▣"V$;:FORZ=1TO8:
    PRINTC(Z);:NEXT:PRINT
450 PRINT"▣▣▣▣▣▣▣";:IF AA=
    1THENPRINT"−";
460 IF AA=2THENPRINT"+";
470 PRINT"□□□□▮▮▮▮▮▮▮"A:PRINT
    "○"V$;
480 FORZ=1TO8:PRINTD(Z);:NEXT Z:ZA=
    INT(SS/16):ZB=SS−(16*ZA)
490 PRINT"□□□□□"MID$(Z$,ZA+1,1);
    MID$(Z$,ZB+1,1): GOTO150
500 I$="":PRINT"▣▣▣▣▣▣▣▣
    ▣▣▣▣▣▣▣▣▣▣▣▣▣
    ▣▣NUMERO(TRA −128 E +127)?□
    □□□□□▮▮▮▮▮▮▮";
510 FOR Z=1 TO 4
520 GETJ$:IFZ=1 AND (J$="−" OR J$=
    "+")THEN U$=J$:PRINTU$;:NEXT Z
530 IF Z=1 THEN 520
540 PRINT "*▮□▮▮";:IF J$="" THEN 520
550 IF J$=CHR$(13) THEN 610
560 IF J$=CHR$(20) THEN 500
570 IF ASC(J$)<48 OR ASC(J$)>57 THEN
    520
580 I$=I$+J$:PRINT J$;:NEXT Z
```

```
590 GET J$:IF J$=CHR$(20) THEN 500
600 IF J$<>CHR$(13) THEN 590
610 IF VAL(I$)<0 OR (VAL(I$)>128ANDU$
    ="−") OR (VAL(I$)>127ANDU$="
    +") THEN500
620 IF U$="−"THEN AA=2:GOTO640
630 AA=1
640 PRINT:PRINT"■□□□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□";:A=VAL(I$):
    GOTO280
```

```
20 POKE 36878,15
30 FOR Z=1 TO 8:READ A(Z): NEXT Z:
   DATA 128,64,32,16,8,4,2,1
40 Z$="0123456789ABCDEF":A=0:A$=
   "+−":AA=1:POKE 650,128:V$="■
   ■■"
50 POKE 36879,25:PRINT"■■■■■
   ■■■"
60 FOR Z=1 TO 9:PRINT "■■■□□□
   □□□□□□□□□□□□□□□":NEXT Z
70 PRINT"■■■■■■
COMPLEMENTO A 2"
80 PRINT "■■■■NUMERI NEGATIVI"
90 PRINT "■■■− − − − − − − − −
   − − − − − − − − −"
100 PRINT "▒DEC□□□□□□□BIN□□□
    □□□ESA"
110 PRINT "■■■■■■■■
    FLIP BITS":PRINT TAB(104)"+1"
120 PRINT "■■■■■+"TAB(19)"+"
130 PRINT "■■■■■■■■− −
    − − − − − − − − − − − − − − − −
    − − − − − −";
140 PRINT "0□0□0□0□0□0□0□0□0□
    00":GOTO 300
150 GET K$:IF K$="" THEN 150
160 S=0:SS=0:IF K$<>"□" AND K$<
    >"B" THEN 500
170 A1=A:IF K$<>"□" THEN 210
180 IF AA=2 THEN A1=A1−1
190 IF AA=1 THEN A1=A1+1
200 GOTO 240
210 IF K$<>"B" THEN 240
220 IF AA=2 THEN A1=A1+1
230 IF AA=1 THEN A1=A1−1
240 IF A1<0 OR A1>128 THEN AA=AA
    +1
250 IF A1>128 THEN A1=127
260 IF A1<0 THEN A1=1
270 A=A1
280 IF A=0 THEN AA=1
290 IF A=128 THEN AA=2:POKE
    36876,200:FOR Z=1 TO 20:NEXT Z:POKE
    36876,0
300 IF AA>2 THEN AA=1
310 W=0:IF AA=1 THEN T=1:TT=0
320 IF AA=2 THEN T=0:TT=1:W=1
330 A1=A−W:FOR Z=1 TO 8:IF A(Z)<=
    A1 THEN B(Z)=T:C(Z)=TT:A1=A1−A
    (Z):GOTO 350
340 B(Z)=TT:C(Z)=T
350 IF B(Z)=1 THEN S=S+A(Z)
360 NEXT Z:A2=0:FOR Z=1 TO 8:IF C(Z)
    =0 THEN A2=A2+A(Z)
370 NEXT Z:A1=(255−A2)+T:IF A1>
    255 THEN A1=0
380 FOR Z=1TO8:IFA(Z)<=A1+
    W THEN D(Z)=1:A1=A1−A(Z):SS=SS
    +A(Z):GOTO400
390 D(Z)=0
400 NEXT Z
410 PRINT "■■■■■■■"MID$(A$,
    AA,1):PRINT "■■■■□□□□■
    ■■■"RIGHT$(STR$(A),LEN(STR$
    (A))−1)
420 PRINT "■■■■"V$;:FOR Z=
    1 TO 8:PRINT RIGHT$(STR$(B(Z)),1)"□";:
    NEXT Z
430 ZA=INT(S/16):ZB=S−(16*ZA):
    PRINT "■"MID$(Z$,ZA+1,1)MID$(Z$,ZB
    +1,1)
440 PRINT "■■■■■"V$;:FOR Z=
    1 TO 8:PRINT RIGHT$(STR$(C(Z)),1)"■";:
    NEXT Z:PRINT
450 PRINT "■■■■■■■";: IF AA
    =1 THEN PRINT "−"
460 IF AA=2 THEN PRINT "+"
470 PRINT "□□□□■■■■■"
    RIGHT$(STR$(A),LEN(STR$(A))−1):
    PRINT "■"V$;
480 FOR Z=1 TO 8:PRINT RIGHT$(STR$(D
    (Z)),1)"□";:NEXT Z:ZA=INT(SS/16):ZB=
    SS−(16*ZA)
490 PRINT "■"MID$(Z$,ZA+1,1)MID$(Z$,
    ZB+1,1):GOTO 150
500 I$="":PRINT "■■■■■■■
    ■■■■■■■■■■■■■■
    ■■NUMERO?▒(TRA −128■E▒
    +127)>□□□□■■■■";
510 FOR Z=1 TO 4
520 GET J$:IF Z=1 AND (J$="−" OR J$
    ="+")THEN U$=J$:PRINT U$;: NEXT Z
540 PRINT "*■□■";:IF J$="" OR Z=1
    THEN 520
550 IF J$=CHR$(13) THEN 610
560 IF J$=CHR$(20) THEN 500
570 IF ASC(J$)<48 OR ASC(J$)>57 THEN
    520
580 I$=I$+J$:PRINTJ$;:NEXT Z
590 GET J$:IF J$=CHR$(20) THEN 500
600 IF J$<>CHR$(13) THEN 590
610 IF VAL(I$)<0 OR(VAL(I$)>
    128 AND U$="−")OR(VAL(I$)>
    127 AND U$="+") THEN 500
620 IF U$="−"THEN AA=2: GOTO 640
630 AA=1
640 PRINT:PRINT "■■□□□□□□□□
    □□□□□□□□□□□□□□□□□□
    □□";:A=VAL(I$):GOTO280
```

```
10 MODE 1
20 VDU 23;8202;0;0;0;
30 VDU 19,1,6,0,0,0,0,0
40 PRINTTAB(13,3)"NUMERI□□NEGATIVI"
50 PRINTTAB(13,4)STRING$(16,CHR$(224))
   TAB(6)"Dec"TAB(19)"Bin"TAB(31)"Esa"
60 PRINTTAB(8,13)"+"TAB(33,13)"+"
70 PRINTTAB(11,18)"Complemento a 2"
80 PRINTTAB(8,22)"0□□□0□0□0□0□□
   0□0□0□0□□□□□00"
90 PRINTTAB(7,27)"PREMI UNO SPAZIO PER
   INCREMENTARE"
100 PRINTTAB(7,29)"PREMI UNA B PER
    DECREMENTARE"
110 GCOL0,1
120 MOVE320,480:MOVE320,704:PLOT
    85,960,480:PLOT85,960,704
130 PRINTTAB(12,11)"COMPLEMENTO"
140 PRINTTAB(26,15)"+1"
150 VDU 31,6,21,224,224,224,31,31,21,224,
    224,224
160 PRINTTAB(11,21)STRING$(18,CHR$(224))
170 ?&70=0
180 T=?&70:IF T=128 THEN SOUND1,
    −15,100,10
190 PROCBIN(8): PROCHEX(8)
200 T=T+256*(T>127):PROCDEC(8)
210 T=255−T:PROCBIN(13)
220 T=T+1:PROCBIN(20):PROCHEX(20)
230 T=T+256*(T>127):PROCDEC(20)
240 *FX21,0
250 G=GET
260 IF G=32 THEN?&70=?&70+1:
    GOTO 180
270 IF G=66 THEN?&70=?&70−1:
    GOTO 180
280 PRINTTAB(0,30);:INPUT?&70:PRINTTAB
    (0,30)STRING$(39,"□");:GOTO180
290 DEF PROCBIN(Y)
300 FOR X=0 TO 7
310 IF−(T AND 2∧X) THEN PRINT TAB(27
    −X*2+(X>3),Y)"1" ELSE PRINT TAB
    (27−X*2+(X>3),Y)"0"
320 NEXT X
330 ENDPROC
340 DEF PROCHEX(Y)
350 X=(T AND 240)/16
360 A$=CHR$(X+48−7*(X>9))
370 X=(T AND 15)
380 B$=CHR$(X+48−7*(X>9))
390 PRINTTAB(32,Y);A$+B$
400 ENDPROC
410 DEF PROCDEC(Y)
```

```
420 PRINT TAB(6 – LEN(STR$(T)),Y);"□□□"
    T:ENDPROC
```

```
10 CLS
20 PRINT@8,"NUMERI NEGATIVI";
30 PRINT@40,STRING$(16,CHR$(131));
40 PRINT@65,"DEC"TAB(15)"BIN"TAB(28)
   "ESA"
50 PRINT@226,"+"TAB(29)"+";
60 PRINT@361,"COMPLEMENTO A 2"
70 PRINT@483,"Ø□□□□□Ø□Ø□Ø□Ø□□Ø
   □Ø□Ø□Ø□□□□□□ØØ";
80 FOR J=1474 TO 1502:POKE J,131:NEXT
90 FOR J=1 TO 7
100 FOR K=1 TO 24
110 POKE 1123+K+32*J,175
120 NEXT K,J
130 PRINT@165,"COMPLEMENTO";
140 PRINT@313,"+1";
150 AT=AT AND 255
160 T=AT:IF T=128 THEN SOUND 3Ø,2
170 LN=3:GOSUB28Ø:GOSUB31Ø
180 T=T+256*(T>127):GOSUB34Ø
190 T=255–T:LN=7:GOSUB28Ø
200 T=T+1:T=T AND 255:LN=13:GOSUB
    28Ø:GOSUB31Ø
210 T=T+256*(T>128):GOSUB34Ø
220 IN$=INKEY$:IFIN$< >"B" AND IN$<
    >"□" AND IN$< >CHR$
    (13) THEN 22Ø
230 IF IN$="B" THENAT=AT–1:GOTO15Ø
240 IF IN$="□" THEN AT=AT+1:GOTO
    15Ø
250 PRINT@384,;:INPUT AT
260 PRINT@384,"□□□□□";
270 GOTO 15Ø
280 FOR X=Ø to 7
290 IF–(T AND 2↑X) THENPRINT@LN*32
    +23–X*2+(X>3),"1"; ELSE PRINT@
    LN*32+23–X*2+(X>3),"Ø";
300 NEXT:RETURN
310 IF T<16 THENA$="Ø" ELSE A$=""
320 PRINT@LN*32+29,A$+HEX$(T);
330 RETURN
340 PRINT@32*LN,MID$("□□□"+STR$(T),
    LEN(STR$(T)));
350 RETURN
```

LA CONVENZIONE SUL SEGNO

Per molti scopi, come si è detto, un numero binario o hex corrisponde perfettamente a un numero sia positivo che negativo. Talvolta però si vuole conoscere se un numero è positivo o negativo.

I salti, nei programmi in codice macchina, richiedono che si specifichi il numero dei byte del salto, in valori positivi per un salto in avanti, in negativi se all'indietro. Il computer considera il primo bit del numero binario e da solo valuta se questo sia negativo o positivo. Se il primo bit è 1 il computer lo assume come negativo, se Ø come positivo.

Nella convenzione sul segno, 128 corrisponde al suo negativo e il computer riprende a contare in avanti

Questo procedimento segue una *convenzione sul segno*. Ciò significa che il computer, invece di far valere i numeri binari di otto bit da Ø a 255, li considera da –128 a +127. Nel programma di conversione in complemento 2, si sarà udito il segnale 'bip' quando si arriva a 128. Ciò avviene perché 128, cioè 1ØØØØØØØ in binario, o 8Ø in hex, corrisponde al suo negativo. (Proviamo: –1ØØØØØØØ+1ØØØØØØØ= (1) ØØØØØØØØ o Ø in binario di otto bit; 8Ø+8Ø= (1) ØØ o Ø in hex in due cifre; 128–128=Ø in decimale.)

Cosa succede allora? Dato che 1ØØØØØØØ ha 1 nel suo primo bit, il computer lo considera numero negativo: –128. D'altra parte zero, o ØØØØØØØØ, ha Ø nel suo primo bit e il computer lo considera positivo.

UNA GRAFICA PIÙ SOFISTICATA!

Gli home computer offrono molte possibilità all'artista in erba. Ecco alcuni metodi per estendere l'uso dei comandi BASIC per la grafica e per creare nuovi disegni su schermo.

Avendo acquisito le basi del disegno sullo schermo, si può cominciare ad allargare il proprio impegno artistico ricorrendo ad alcuni speciali comandi per la grafica messi a disposizione dal computer. Comandi quali MOVE, PLOT, DRAW, PAINT e CIRCLE liberano l'immaginazione e permettono di creare qualsiasi cosa, da un grafico per illustrare un programma finanziario, allo scenario per un eccitante gioco d'avventura.

A pagina 84 abbiamo visto l'uso di comandi per creare disegni sullo schermo mediante linee e, in alcuni casi, per aggiungere il colore. Queste tecniche di base si possono estendere al disegno di punti, triangoli, quadrati e cerchi: usate da sole, o in combinazione con i colori, possono diventare un mezzo efficace per creare immagini statiche o dinamiche.

L'uso di questi comandi varia da computer a computer, ma è sempre possibile ottenere qualche effetto di grande interesse. Le eccezioni sono il Commodore 64 e il Vic 20, il cui BASIC standard non possiede tali comandi. Si possono però aggiungere, acquistando una cartuccia ROM (vedere pagina 87) e questa lezione contiene un paragrafo dedicato all'uso del Simon's BASIC, che facilita l'accesso alle capacità grafiche del Commodore.

CERCHI ED ARCHI

Cerchi e archi sono tra gli "strumenti" più utili dello Spectrum per produrre una grafica statica su schermo.

In questo programma per disegnare un campo da golf, vengono usati per riprodurre alberi, recinti, zone d'acqua e d'erba, ostacoli.

Possiamo verificare i nostri progressi nel corso della trascrizione, eseguendo un gruppo di linee alla volta. Se si evita di digitare NEW, alla fine si ottiene il paesaggio della **figura 1**.

Prima di iniziare con i cerchi, però, conviene disegnare la casetta sullo sfondo (il "circolo del golf"):

```
90 BORDER 4: PAPER 4:CLS
200 LET w=10: LET s=50
210 FOR c=162 TO 174
220 PLOT INK 2;w,c
230 DRAW INK 2;s,0
240 LET w=w+2: LET s=s-4
250 NEXT c
260 FOR b=148 TO 162
270 PLOT INK 2;10,b
280 DRAW INK 2;50,0
290 NEXT b
295 DRAW INK 2;10,-3: DRAW 0,-11
300 PRINT INK 0;AT 2,2; "█"; AT 2,4;
    "█"; AT 2,6; "█"
```

Le linee da 200 a 250 tracciano il tetto, con tecniche simili a quella già viste nella precedente lezione (pagine 84-86). Il disegno inizia dalla posizione 10, 162, con una linea larga 50 pixel, poi la larghezza diminuisce di quattro pixel, poi la larghezza diminuisce di quattro pixel per ogni pixel in altezza. Un ciclo simile, dalla linea 260 alla 290, disegna le pareti e la linea 295 disegna il portico. La linea 300 si occupa delle finestre con il metodo più semplice: adoperando un quadrato nero tratto dai caratteri grafici ROM.

IL DISEGNO DI UN ARCO

Sullo Spectrum, come si è già spiegato (pagina 86) il modo più semplice di disegnare un cerchio completo è quello di usare il comando CIRCLE. Se però si vuole soltanto una porzione di cerchio, allora è più semplice impiegare il comando DRAW. Modificando opportunamente questo comando, si ottiene una grande varietà di effetti, perciò vale la pena di fare qualche esperimento prima di procedere. Si provi per esempio:

```
10 PLOT 130, 30
20 DRAW 0, 10, 1
30 GOTO 20
```

(Non si tenga conto del messeaggio d'errore). Come si ricorderà, i primi due numeri alla linea 20 fanno disegnare al computer una linea dal punto di PLOT a un altro 10 pixel più in alto. L'ultimo valore, nel comando DRAW, serve per disegnare una linea curva, anziché retta: la curvatura è determinata dalla grandezza del numero. È così che lo Spectrum disegna sezioni di cerchio. Un cerchio intero è rap-

presentanto da 2 volte π, quindi 1 genera circa un sesto di cerchio (o, più precisamente, 1 diviso 2 volte π).

Se si cambia la linea 20 in:

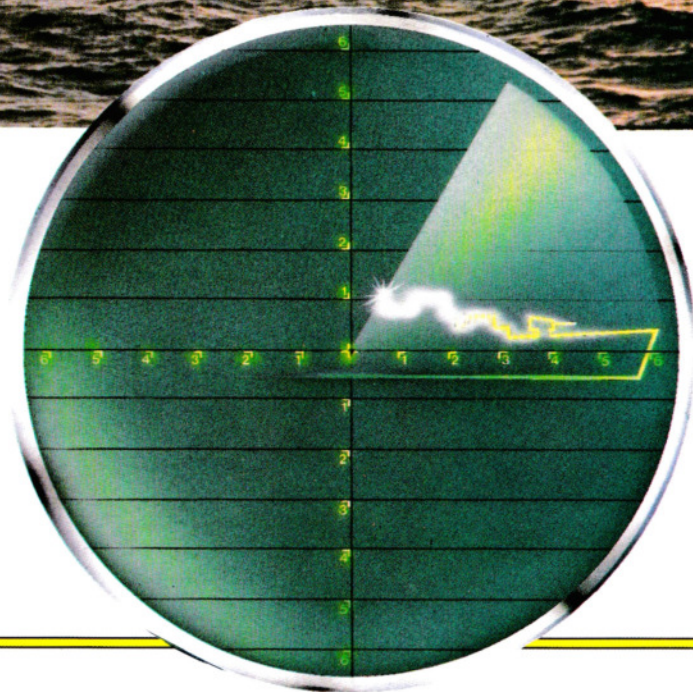
```
20 DRAW 0, 10, 2
```

si vedrà che il risultato è una serie di curve più pronunciate. Analogamente: 0, 10, 3 crea un disegno dentellato, 0, 10, 4 un pezzo di recinto a catenella (o mezzo tronco di palma, a seconda di come lo si guarda!), mentre 0, 10, 6 produce una spirale.

Se invece si prova:.

```
20 DRAW 0, 10, 2*PI
```

si può avere una sorpresa: lo Spectrum cerca di disegnare un cerchio che ha due

- USO DI LINEE MOBILI SUL DRAGON E SULL'ACORN
- UN CAMPO DA GOLF PER LO SPECTRUM E PER L'ACORN
- LETTERE SUL DRAGON
- NUOVI EFFETTI DI COLORI
- DISEGNI FATTI CON CERCHI E ARCHI
- IL SIMON'S BASIC PER LA GRAFICA DEL COMMODORE

punti in linea retta. Ma dato che un simile cerchio sarebbe un bel po' più grande della nostra stanza (in realtà anche più grande del sistema solare), esso disegna solo la parte che può.

Per disegnare il paesaggio si digiti:

```
10 PLOT 0, 100
20 DRAW RND*5+5, 0, 2
30 GOTO 20
```

Questa sezione disegna la forma di un'onda di un mare agitato: per acque più calme, si provi RND* × 10 + 10 o anche RND* × 15 + 10.

Va ricordato, nel disegno di simili archi, che un numero negativo in fondo a un comando DRAW produce un'immagine riflessa dell'arco. Ora si cancellino le line e da 10 a 30, prima di continuare nel disegno del campo da golf.

RECINTO E LAGO

Nel programma del campo di golf ci sono due esempi di archi usati per effetti grafici: un piccolo recinto davanti alla sede del circolo e il lago.

Si scrivano intanto queste linee:

```
310 FOR f=0 TO 84 STEP 3
320 PLOT f,142
330 DRAW 3,0,-3
340 NEXT f
345 DRAW 35,-42: DRAW -12,-6
```

Il punto di partenza sullo schermo è 0, 142. La linea 330 disegna una fitta serie di minuscoli archi (dei semierchi larghi tre pixel), che formano il recinto: il loro numero è stabilito dal ciclo FOR ... NEXT.

Adesso si trascriva e si esegua:

```
100 LET x=130: LET y=125: LET z=50
110 PLOT INK 5;x,0
120 DRAW INK 5;y,z,-1.25
130 LET x=x+1: LET y=y-1: LET z=z
    -1
140 IF x>254 THEN GOTO 170
150 IF z<1 THEN LET z=0
160 GOTO 110
```

Qui il disegno è più complesso. Prima il computer si posiziona in un punto 130 pixel da sinistra e 0 dal basso dello schermo, poi traccia una linea fino a un punto 125 pixel a destra e alto 50 pixel dal mar-

gine in basso. Contemporaneamente, viene operata una curvatura di −1,25 radianti.

A questo punto entrano in gioco le variabili. La X fa partire ogni linea un pixel più a destra, la Y un pixel in meno della precedente (altrimenti uscirebbe dallo schermo), mentre la Z fa finire la linea un pixel sotto quella precedente.

Naturalmente, alla fine Z assume un valore negativo, e il computer continuerebbe a stampare (invano) fuori dello schermo: da qui la necessità della linea 150, che limita il tracciamento delle linette al margine inferiore dello schermo.

ALBERI E CESPUGLI

Per creare alberi e arbusti, il programma del campo di golf adopera dei cerchi completi (non essendo disponibile sullo Spectrum un comando di miglior effetto, tipo PAINT). Queste linee disporranno a caso qualche cespuglio sul campo:

```
400 FOR r=172 TO 168 STEP -1
410 LET x=RND*45+195
415 PLOT x,r-2: DRAW 0,-2
420 CIRCLE x,r,RND*2+1
440 CIRCLE x+10,r,RND*2+1
450 NEXT r
```

Queste linee, invece, visualizzano arbusti simili sul lato destro:

```
460 FOR r=135 TO 172 STEP 6
470 LET y=252
480 CIRCLE y,r,RND+2
490 NEXT r
```

Gli alberi nell'angolo inferiore a sinistra, sono troppo grandi per essere disegnati a caso. Si usa perciò una tecnica diversa, con READ ... DATA (si veda alle pagine 104-109):

```
900 FOR w=1 TO 3
910 READ a,b
920 PLOT a,b
930 DRAW 0,-24
940 LET f=RND*5+5
950 CIRCLE a-10,b+f,f: CIRCLE a,b+f,
   f: CIRCLE a+10,b+f,f
960 CIRCLE a-5,b+f*2,f: CIRCLE a+5,b+
   f*2,f
970 CIRCLE a,b+f*3,f
980 NEXT w
3000 DATA 20,70,52,85,84,100
```

Il trucco consiste nel disegnare prima i tronchi spostandosi verso il basso dai punti di partenza di PLOT per non avere antiestetici pezzi di tronco sovrapposti alle foglie. I tronchi iniziano in 20, 70 e così via grazie alle DATA nella linea 3000. La linea 940 sceglie a caso la grandezza del fogliame, e b+f (linea 950) garantisce che i cerchi si trovino a una giusta distanza dai tronchi.

RITOCCHI FINALI

La pedana di tiro (qui è più simile a un tappetino di gomma!) viene disegnata da queste linee:

```
1000 LET t=30
1010 FOR y=0 TO 10
1020 PLOT t,y
1030 DRAW -30,30
1040 LET t=t+2
1050 NEXT y
```

Queste altre linee visualizzano le bandierine sul prato:

```
170 PLOT 220,140
180 DRAW 0,15: DRAW 8,-3: DRAW -8,
   -2
190 PLOT 22, 120
195 DRAW 0,18: DRAW 9,-3: DRAW -9,
   -2
```

Infine, mettiamoci degli ostacoli. Il metodo più preciso, ma più lento, per disegnarli (in assenza del comando PAINT), è partire con una piccola ellisse e poi ingrandirla, pixel dopo pixel, fino alla dimensione giusta. Il disegno di ellissi viene spiegato dettagliatamente in una lezione sulle funzioni matematiche. Per adesso, si trascrivano queste linee (dopo il RUN ci sarà il tempo per prendersi un caffè!):

```
1495 LET r=1
1500 FOR x=0 TO 2*PI STEP PI/180
1510 PLOT INK 6;168+r*SIN x,147+r*
   COS x/2.5
1520 PLOT INK 6;235+r*SIN x,106+
   r*COS x/2.75
1530 PLOT INK 6;225+r*SIN x,97+
   r*COS x/2.5
1540 NEXT X
1550 LET r=r+2
1560 IF r>20 THEN GOTO 6000
1570 GOTO 1500
```

In alternativa si possono fare ostacoli con i più semplici, ma più rozzi, UDG.

Con la cartuccia aggiuntiva "Simon's BASIC" il Commodore si arricchisce di una serie di comandi supplementari, alcuni dei quali con la funzione di semplificare la programmazione della grafica, come in parte si è già visto (pagine 87-88).

ARC e ANGL sono due comandi per il disegno che non abbiamo ancora incontrato; il primo disegna archi di cerchio e ha questo formato:

```
99 ARC 150,50,60,270,1,30,30,1
```

Abitualmente va preceduto dal comando HIRES, per cui si scriva anche questa linea:

```
90 HIRES 0,1
```

I primi due numeri dopo il comando ARC definiscono le coordinate sullo schermo del centro del cerchio di cui si richiede l'arco. Come al solito, questi valori si riferiscono alle posizioni in pixel in notazione standard: il primo è il valore orizzontale (X), il secondo è quello verticale (Y).

Si provi a cambiare questi valori, lanciando varie volte il programma: si vedrà che, all'aumentare di ciascuno di questi valori, la linea viene tracciata sempre più vicina alla parte inferiore destra dello schermo, fino a toccare il margine.

L'effettiva lunghezza dell'arco è regolata dalle due successive coppie di valori. La prima (60 e 270) rappresentano gli *angoli* iniziale e finale. Immaginiamo che si tratti del quadrante di un orologio: la misura dell'angolo inizia e finisce alle 12, con i valori 0 e 360. Cosicché, la curva dell'esempio comincia alle 2 e finisce alle 9. Si cambino di nuovo i valori di questi numeri per valutare le conseguenze.

Il numero successivo è *l'incremento di curvatura* e, aggiustandone il valore nell'intervallo da 1 a 360, si modifica, appunto, la curvatura: con il valore 10, l'aspetto della curva è più grossolano di quella del programma iniziale. Per farsi un'idea delle possibilità, si apportino queste correzioni al programma originale:

```
90 HIRES 0,1: FOR N=10 TO 360 STEP 10
99 ARC 160,80,0,360,N,84,60,1
100 NEXT: PAUSE 10
```

Qui si può osservare l'effetto di una correzione dell'incremento della curvatura a passi di 10 gradi su una curva che parte e termina nello stesso punto, ossia un cerchio. (L'uso della variabile N, in questo esempio, sottolinea la possibilità di operare modifiche col minimo apporto di programmazione).

La coppia di valori che segue N regola l'aspetto fisico del cerchio di cui si visualizza l'arco: il primo valore è la lunghezza in pixel di X (il raggio orizzontale), il secondo è la lunghezza di Y (il raggio verticale). Volendo produrre un arco di cerchio, il valore X deve essere 1, 4 volte il valore di Y nel modo HIRES e 1,6 volte nel modo MULTI. Altrimenti, otterremmo una curva ellittica anziché circolare.

Si provi a cambiare questa coppia di valori, rammentandosi che la massima risoluzione è di 320 pixel in orizzontale e di 200 in verticale nel modo HIRES e metà della risoluzione orizzontale nel modo MULTI.

La somma del primo valore dopo il comando ARC, ossia della coordinata X del centro della curva, e il valore del raggio X della curva non devono superare la larghezza massima dello schermo. Lo stesso vale per l'asse Y. In ambedue i casi, superando le dimensioni massime, la curva *esce* dallo schermo, per riapparire qualora i valori tornino al disotto dei limiti.

L'ultimo numero dell'istruzione ARC seleziona il tipo di tracciamento: specificando uno Ø, il punto in questione si *spegne*, il valore 1 provoca l'*accensione* del punto, mentre un 2 *inverte* la condizione del punto (lo accende se era spento e viceversa). Si provi a usare il valore 2 nel programma, per vedere l'effetto che si ottiene.

IL DISEGNO DI RAGGI

Il comando seguente, ANGL, è usato per disegnare il raggio di un cerchio e può essere utilizzato in più maniere per rappresentare oggetti come ruote o ventagli. Ecco come impiegarlo (ricordarsi di impartire un NEW):

```
10 HIRES Ø,1: FOR N=Ø TO 360 STEP 4
20 ANGL 16Ø,8Ø,N,84,6Ø,1
30 NEXT:PAUSE 1Ø
```

Per risparmiare fatica, anche stavolta viene usato un ciclo FOR ... NEXT per variare progressivamente la variabile N. I primi 2 valori, dopo il comando ANGL, sono le coordinate X e Y del centro del cerchio di cui si vuol disegnare il raggio. Ovviamente, il centro è anche il punto di partenza del raggio. Il valore N modifica l'inclinazione del raggio tracciato. Un valore di 45, ad esempio, disegna una linea dal punto di partenza a un altro posto a 90° rispetto a esso. Si possono usare, come si vede, valori da Ø a 36. Si provi a cambiare il valore di STEP da 1 a 1Ø oppure ad assegnare un valore fisso alla N nella linea 2Ø.

I restanti tre valori hanno la stessa funzione di quelli visti nel comando ARC, ma stavolta è da notare l'utilità del valore Ø in fondo alla linea. Si aggiungano al programma queste linee:

```
30 NEXT
35 CIRCLE 16Ø,8Ø,84,6Ø,1
40 FOR N=360 TO Ø STEP -4
45 ANGL 16Ø,8Ø,N,84,6Ø,Ø
50 NEXT: PAUSE 1Ø
```

Il disegno viene ripetuto, viene aggiunto un cerchio, dopodiché la linea 45 si occupa di cancellare il lavoro fatto dalla 2Ø.

BLOCCHI DI COLORE

Il comando BLOCK permette di creare rettangoli colorati. Si potrebbero usare anche REC e PAINT, ma BLOCK è spesso più conveniente, specie se si vogliono svariati rettangoli. Il suo formato è il seguente (prima si digiti NEW):

```
10 HIRES Ø,1: MULTI 2,5,6
20 BLOCK 1Ø,1Ø,3Ø,3Ø,1
50 PAUSE 1Ø
```

Dopo il RUN, compare un rettangolo di un colore. I primi due valori definiscono l'angolo superiore a sinistra del rettangolo, i

due successivi, invece, l'angolo inferiore a destra. L'ultimo numero è il tipo di tracciamento, che nel modo MULTI (selezionato alla linea precedente) sceglie il primo colore in opzione: il 2 corrisponde al rosso. Si provi a sostituire il valore del tipo di tracciamento con 2 o 3 e si riesegua il programma. O, meglio ancora, si aggiunga:

```
30 BLOCK 20,20,50,50,2
40 BLOCK 30,30,70,70,3
```

Dopo il RUN, compariranno tre blocchi colorati. Per un risultato più interessante si apportino queste correzioni che servono a visualizzare una sequenza casuale di quindici blocchi rossi, verdi e blu:

```
20 N=5: M=10: C=1
30 BLOCK N,N,M,M,C
40 N=N+5: M=M+10: C=INT(RND(1)*3
   +1)
50 IF M<160 THEN GOTO 30
60 PAUSE 10
```

DISEGNO

DRAW e ROT sono due comandi usati congiuntamente per visualizzare una forma precisa. Si usa una sequenza di istruzioni numerate secondo il seguente formato (eseguire prima un NEW):

```
100 DRAW A$,150,150,1
```

La componente A$ dell'istruzione è quella che definisce la forma del disegno, la coppia di valori che seguono sono le coordinate X e Y del punto di partenza e l'ultimo numero finale è il solito valore del tipo di plot. A$ può contenere numeri da Ø a 9 il cui significato è:

Ø movimento di un pixel a destra
1 movimento di un pixel in alto
2 movimento di un pixel in basso
3 movimento di un pixel a sinistra
4 non valido
5 movimento a destra e visualizza un pixel
6 movimento in alto e visualizza un pixel
7 movimento in basso e visualizza un pixel
8 movimento a sinistra e visualizza un pixel
9 fine disegno

Si può usare ogni combinazione di numeri nella stringa (A$) con un massimo di 74 in ogni singola linea (ciò per lasciar spazio agli altri parametri del comando DRAW). Disegni elaborati richiedono un maggior numero di istruzioni di quelle contenibili in una sola linea, ma si può risolvere il problema concatenando diverse stringhe fino a una lunghezza massima di 255 caratteri. Un esempio è:

```
A$="UNA STRINGA DI 74 CARATTERI"
A$=A$+"ANCORA PIÙ CARATTERI"
```

che allunga notevolmente A$! Ora vediamo un esempio pratico per A$:

```
10 A$="5555555555555555555657565755
   555555506016378888888888888888888
   88888888"
20 A$=A$+"100000000000000555068815
   550688655188883655551888883655555
   50063"
30 A$=A$+"888888888888888881105555
   555555755555063888886555550688888
   88865"
40 A$=A$+"55555500068838105555188
   9"
```

Queste poche righe definiscono uno sciatore in discesa! Ma per visualizzare A$ serve ancora un pezzo di programma ed ecco che entra in gioco il comando ROT.

ROT permette di specificare orientamento e dimensione della forma definita con il comando DRAW.

Si aggiunga questa linea:

```
200 ROT 0,1
```

Il primo parametro, dopo il comando, stabilisce il grado di rotazione del disegno. I seguenti valori corrispondono ad ogni possibile orientamento:

Ø Ø gradi rotazione
1 45 gradi di rotazione
2 9Ø gradi di rotazione
3 135 gradi di rotazione
4 18Ø gradi di rotazione
5 225 gradi di rotazione
6 27Ø gradi di rotazione
7 315 gradi di rotazione

Una volta terminato (e memorizzato!) il programma, potremo provare questi valori, sostituendoli nella linea 2ØØ.

Il secondo valore dopo il comando ROT specifica la dimensione del disegno: con il valore 1 la forma è visulizzata a grandezza normale. Aumentando questo numero si ingrandisce il disegno il disegno, ma occorre non oltrepassare l'area disponibile sullo schermo. Per eseguire il RUN, è necesario aggiungere queste linee:

```
50 HIRES 0, 1
250 PAUSE 20
```

Si rilanci più volte il programma, cambiando, qua e là, i vari parametri. Alcuni tentativi andranno a vuoto, ma, con un poco di criterio, si riuscirà ad ottenere una serie di immagini davvero originali.

Per imparare come combinare più comandi del Simon's BASIC, aggiungiamo al programma anche queste linee:

```
50 HIRES 0,1:MULTI 2,5,3:COLOUR 6,1
60 LINE 0,20,320,60,1
65 LINE 32,1,80,7,1
70 LINE 0,10,32,1,1
80 LINE 25,25,100,1,1
85 LINE 100,1,250,50,1
90 CIRCLE 150,20,5,6,1
95 PAINT 5,5,3
115 FOR N=1 TO 5:READ X1,X2,X3,X4,X5
120 FOR Z=1 TO X1 STEP 4
125 LINE X4,X5+Z,X3,(X5-20)-Z,2+
    RND(1)*2
130 LINE X2,X5+Z,X3,(X5-20)+Z,2+
    RND(1)*2:NEXT
140 BLOCK X3,X5+X1-20,X3,X5-X1,1::
    NEXT N:ROT 0,1
200 LOW COL 1,4,2:FOR Z=25 TO 60
    STEP 3:DRAW A$,Z-3,163,1
205 DRAW A$,Z,163,2:NEXT Z
210 DRAW A$,25,169,3
1000 DATA 50,33,40,47,90
1002 DATA 20,5,10,15,30
1004 DATA 25,15,20,25,60
1006 DATA 120,1,10,19,90
1008 DATA 20,85,90,95,30
```

Il Vic 2Ø ha una grafica eccellente, ma per usarla appieno è indispensabile aggiungere una cartuccia Super Expander: questa fornisce tutti gli abituali comandi grafici come DRAW, CIRCLE ecc. Il programma seguente mostra come creare complessi disegni con i nuovi comandi:

```
10 GRAPHIC 2:COLOR 1,6,6,0
20 CIRCLE 2,500,500,100,400
25 FOR Z=1 TO 150 STEP 10
30 CIRCLE 2,500+SIN(Z)*30,500+Z,100,
   400,60,130:NEXT
35 FOR Z=1 TO 20 STEP 7
40 CIRCLE 2,450,440,30-Z,20
45 CIRCLE 2,550,440,30-Z,20
50 DRAW 2,470,740+Z TO 490,720+
   Z TO 510,740+Z TO 530,720+
   Z TO 550,740+Z:NEXT
60 DRAW 2,490,500 TO 470,640 TO 490,660
   TO 510,650
70 CIRCLE 2,510,750,40,20,10,40
80 FOR Z=1 TO 500 STEP20
90 DRAW 2,400,520 TO 200-(Z*.3), 600+Z
   TO 520,1000 TO 800+(Z*.3),600+Z TO
   600,500: NEXT
100 FOR Z=1 TO 2:CHAR 15-Z,8-Z,"*":
    NEXT
110 GOTO 110
```

Attraverso una serie di ellissi, linee curve e rette il programma disegna un volto di donna. Le linee 2Ø e 3Ø disegnano il viso e i capelli, le linee da 35 a 7Ø gli occhi, il naso e la bocca, e le linee da 8 a 16 le spalle. Chi possiede la cartuccia può divertirsi a disegnare il volto di un uomo.

I comandi PLOT e DRAW non si limitano a disegnare semplici linee rette del tipo mostrato a pagina 88. Con i controlli giusti si possono ottenere curve, zig zag e una vasta gamma di trame. Sfruttando anche i colori, si hanno le basi per una gamma completa di effetti visivi.

A pagina 84 si è descritto come produrre semplici disegni mediante PLOT e DRAW e come aggiungervi il colore con altre poche istruzioni. Una volta imparati i comandi fondamentali, possiamo aggiungere una nuova dimensione alla nostra creatività grafica, tracciando linee la cui forma si modifica durante il disegno. Alla superficie di un mare si aggiungono le onde, al profilo di una scogliera le rocce, a un castello i bastioni. E tutto con una sola linea di programma.

Il segreto per aggiungere trame, con i comandi PLOT e DRAW, è di non limitarsi alle coordinate fisse con cui questi definiscono una singola linea, ma di sfruttare la velocità del computer per tracciare molte linee brevi sullo schermo, ognuna a diverse angolature. Invece di una linea a zig zag, se ne possono disegnare 50 o anche 2000 a diverse altezze, ottenendo effetti visivi di gran lunga superiori. Per esempio, il programma potrebbe spostare il cursore su e giù a una distanza casuale, a ogni passo compiuto in senso orizzontale. Il risultato sarebbe una linea frastagliata irregolare, che serve per disegnare rocce o sagome di vetri rotti.

DISEGNO DI LINEE CASUALI

Un esempio di questo tipo di controllo utilizza dei cicli FOR ... NEXT per cambiare in modo casuale, entro limiti prefissati, le coordinate di una linea. Si provino le linee qui sotto, (il simbolo %, che potrebbe anche essere omesso, serve a velocizzare il disegno. Il suo impiego verrà spiegato in una successiva lezione).

```
10 MODE 2
20 GCOL 0,1
30 LET Y%=800
40 FOR X%=0 TO 1279
50 LET Y%=Y%-30
60 IFY%<0 THEN END
70 MODE X%, Y%
80 FOR A%=1 TO 150
90 PLOT 1,RND(20),RND(30)-15
100 NEXT A%,X%
```

Diversamente dalla maggior parte dei computer, gli apparecchi Acorn permettono di usare l'intero schermo TV per testi e disegni. Tuttavia, può accadere che le immagini siano spostate troppo verso l'alto o verso il basso dello schermo, perdendone così una parte, specie in prossimità degli angoli. Fortunatamente, si può rimediare facilmente con il comando *TV.

*TV1 alza il display di una linea, mentre *TV255 lo abbassa di una linea.

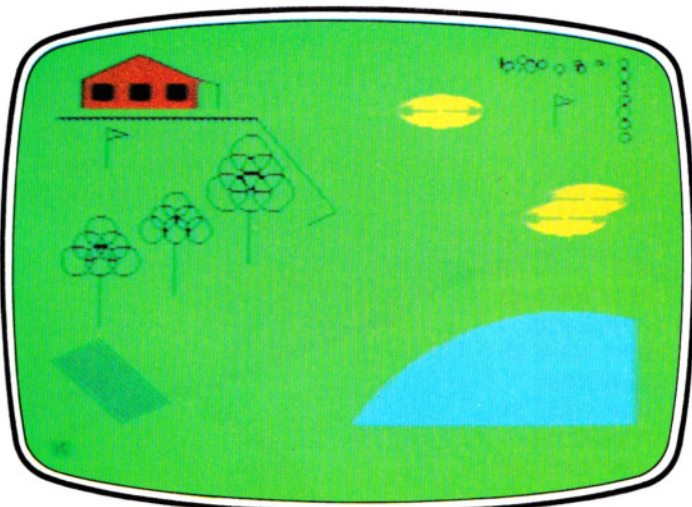
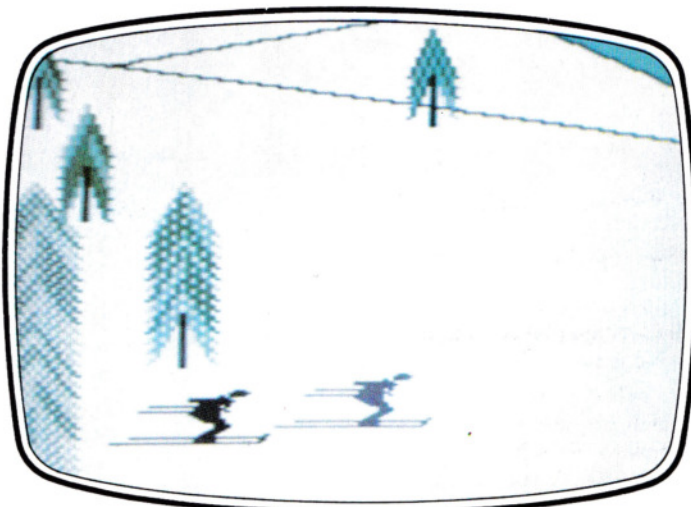
Altri valori, tra 1 e 255, spostano ancora di più l'immagine. Si rammenti, però, che questi comandi hanno effetto soltanto *dopo* la pressione del tasto BREAK o un cambiamento di MODE.

Il risultato è una serie di linee sullo schermo. A questo scopo sono stati selezionati un particolare MODE (linea 10) e un colore con cui disegnare (linea 20).

Fulcro del programma è la linea 90, che disegna una linea da un punto sul margine sinistro dello schermo a un punto casuale a destra, poi da un punto vicino a quest'ultimo a un altro punto a caso e così via. Il punto di partenza viene scelto nelle linee da 30 a 50. Queste linee pongono X%=0 e Y%=770. La linea 90 sposta il cursore al primo punto a caso e lo *accende*. Il successivo punto di partenza è scelto dalle linee 40 e 50, mentre il nuovo punto di arrivo dalla linea 90. L'effetto cambia in base a quante volte viene tracciata ciascuna linea e ai limiti imposti a A% nella linea 80. Le varianti a questa routine sono numerose: si sostituiscano, uno alla volta, i valori dei comandi per vedere il risultato. Si può ottenere un effetto di prospettiva per una catena di montagne, come si vedrà eseguendo il blocco di programma seguente. Si confrontino i valori usati, diversi dai precedenti:

```
10 MODE 2
20 GCOL 0,1
30 LET Y%=-180
40 FOR X%=-10 TO 1279
50 LET Y%=Y%+20
60 IF Y%>150 THEN END
70 MOVE X%,Y%
80 FOR A%=1 TO 150
90 PLOT 1,RND(20),RND(40)-15
100 NEXT A%,X%
```

DISEGNO DI UNA ZONA D'ERBA

Se volessimo soltanto una porzione di schermo intessuto di linee? Il prossimo

1. Un campo da golf per lo Spectrum con archi e cerchi

2. Il Simon's BASIC facilita il disegno sul Commodore

blocco di programma produce una specie di distesa d'erba incolta mista a cespugli (ideale nel caso di un campo di golf, per esempio).

Dopo il NEW, trascriviamo ed eseguiamo queste linee:

```
10 MODE 2
20 GCOL 0,130:GCOL 0,1
30 CLG
40 FOR T=1 TO 50
50 MOVE 600+RND(100),600+RND(100)
60 FOR A=1 TO 10
70 PLOT 1,RND(20),RND(30)-15
80 NEXT A,T
```

La linea 30 crea uno schermo verde, il colore dello sfondo scelto nella linea 20. Stavolta però i punti di partenza sono scelti a caso (linea 50) ogni volta che il programma passa dalla linea 40. Anche i punti d'arrivo sono scelti a caso e disegnati (linea 70) a ogni passaggio dalla linea 60. Il margine dell'area disegnata sarà irregolare, anziché avere una forma precisa e l'effetto può essere variato ampiamente.

Si provi il programma nel MODE 1 poi si riporti la linea 10 al MODE 2.

METTIAMO UN OSTACOLO

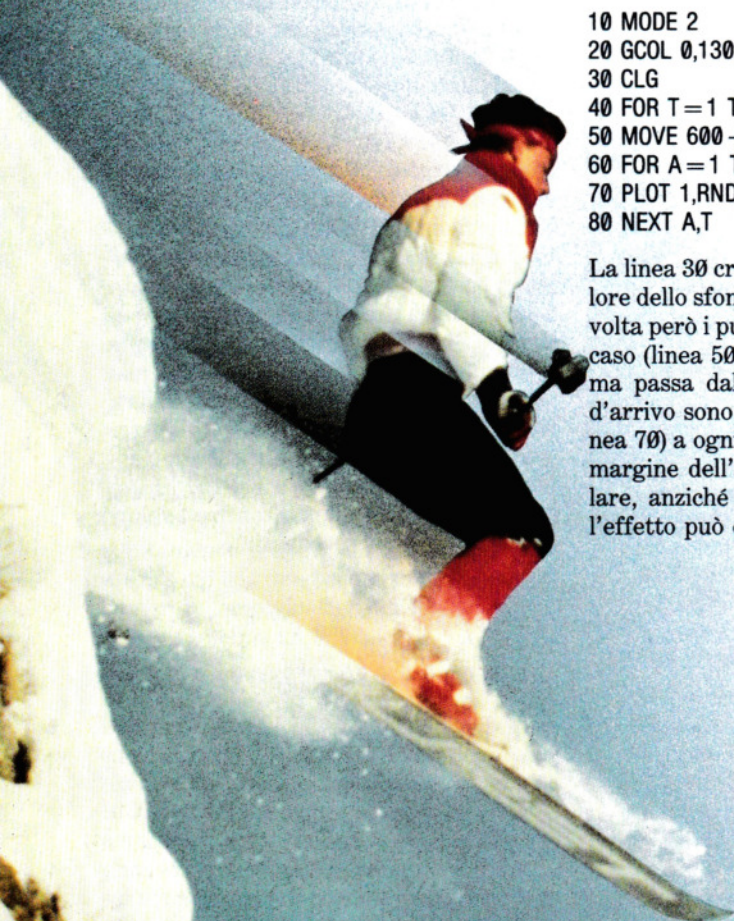
Un elemento adatto, per il campo da golf, potrebbe essere un ostacolo di sabbia. È di nuovo utile la tecnica di porre a caso le coordinate. Si aggiungano queste linee:

```
90 GCOL 0,3
100 FOR T=0 TO 150
120 MOVE 200+RND(100),50+RND(100)
130 FOR A=0 TO 10
140 PLOT 65,20,RND(50)-10
150 NEXT A,T
```

Stavolta il comando PLOT non visualizza linee, ma singoli punti: come per l'erba, viene scelta una posizione di partenza casuale (linea 120). A ogni iterazione del programma nel ciclo FOR ... NEXT, che inizia alla linea 100, viene visualizzato un singolo punto 20 unità in direzione X e da −9 a 40 unità in direzione Y.

Ecco un efficace esempio di questa tecnica puntiforme:

```
200 MODE 2
210 GCOL 0,130
220 CLG
230 GCOL 0,3
240 FOR Y%=1024 TO 100 STEP -8
250 FOR X%=0 TO 1279 STEP RND(5)+10
260 IF RND(1200)<Y%THEN PLOT 69,X%,Y%
270 NEXT X%,Y%
```

La linea 210 seleziona uno sfondo verde, la 260 genera un disegno di ombre punteggiate: è la IF ... THEN alla linea 260 che crea la sorprendente prospettiva di uno sfondo più o meno bianco man mano che il programma procede. Questo tipo di controllo è ideale per un disegno artistico o

per lo scenario di un gioco. Si possono poi mettere insieme più elementi o disporli secondo diversi punti di vista.

Ecco per esempio alcune linee per mettere insieme erba, ostacolo e due alberi:

```
200 GCOL 0,0
210 FOR Y%=1023 TO 850 STEP -10
220 MOVE 0,Y%
230 DRAW 150,Y%
240 NEXT Y%
245 REM......BOSCO N.2
250 FOR Y%=840 TO 700 STEP -10
260 MOVE 0,Y%
270 LET X%=Y%-690
280 DRAW X%-20,Y%
290 NEXT Y%
295 REM......BOSCO N.3
400 LET X1%=500
410 LET X2%=1100
420 FOR Y%=1023 TO 825 STEP -10
430 LET X1%=X1%+10
440 MOVE X1%,Y%
450 LET X2%=X2%-20
460 DRAW X2%,Y%
470 NEXT Y%
```

Digitando queste linee, si cancellano automaticamente quelle del programma precedente.

COMPLETARE IL CAMPO DA GOLF

La seguente sezione trasforma un'area dello sfondo in un lago:

```
500 GCOL 0,4
510 MOVE 1279,650
520 MOVE 1279,600
530 PLOT 85,1100,600
540 PLOT 85,1000,400
550 MOVE 1279,600
560 PLOT 85,1279,100
```

Questo blocco disegna e riempie tre triangoli, il primo preparato dalle linee 510, 520 e 530, il secondo dalle linee 520, 530 e 540. Il terzo, infine, dalle linee 540, 550 e 560.

Con la seguente sezione, l'ultima, si completa il campo da golf con la bandiera per segnare una buca e un rettangolo per simboleggiare una casetta:

```
590 REM......COSTRUZIONE
595 GCOL 0,1
600 MOVE 0,500
610 MOVE 50,500
620 PLOT 85,0,300
630 PLOT 85,50,300
640 REM......BANDIERA
700 MOVE 300,800
710 DRAW 300,900
720 GCOL 0,6
730 MOVE 380,880
740 PLOT 85,300,860
```

Ambedue gli oggetti sono formati da triangoli, nello stesso modo di prima. Disegnare il campo da golf in più fasi consente di provare e rifinire ogni singolo elemento, prima di aggiungerlo al quadro complessivo.

Ora che sappiamo come fare, possiamo creare un nuovo disegno, suggerito dalla nostra fantasia.

IL COMANDO DRAW

I comandi LINE e CIRCLE sono molto utili per disegnare forme semplici e regolari con poca fatica, ma di fronte a forme più sofisticate i programmi divengono eccessivamente lunghi e complessi.

Quanto sarebbe lungo il programma per disegnare la nave della figura a pagina 185, se dovessimo usare un'istruzione LINE per ogni singola linea?

Per evitare il problema, conviene adoperare il comando DRAW, che modifica direttamente il percorso di una linea mentre viene tracciata. Con un solo comando, la linea può essere spostata in più direzioni sullo schermo, rendendo il programma molto più compatto. Le direttive sono contenute, infatti, in una stringa.

Come esempio, digitiamo ed eseguiamo questo programma, che, con sole dieci linee, disegna la nave:

```
10 PMODE 4,1
20 PCLS5
30 SCREEN1,1
40 DRAW"BM23,96C0"
50 DRAW"R28E2U3L6UR6E2R5F2D3R3U2R7
   D2R4U9E2R4F2D5R3U2R4U6E3R5D8"
60 DRAW"R3U24L3UR8DL4D24RF2R5D4R4
   U4R6U9E3R5D10R4U2R4U14RD10R3U2"
70 DRAW"R4D11R7U3E2R4F2D3RD8R3U5E2
   R8D2R6U2R7UE2R6F2D4R4U4E2R5F2"
80 DRAW"R6DL6D3R24G12L195H4U5"
90 PAINT(127,100),0,0
100 GOTO 100
```

Le linee da 10 a 30 stabiliscono le condizioni iniziali. La linea 10 selezioni il PMODE 4 per usare la risoluzione più alta. PCLS 5 alla linea 20 pulisce lo schermo e lo colora di marrone chiaro.

La serie di comandi DRAW (linee da 40 a 80) forma il profilo della nave. I valori necessari al comando DRAW sono quelli racchiusi tra virgolette.

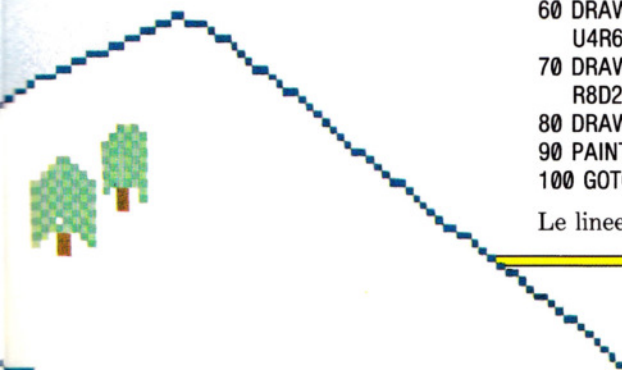
La linea 40 è la più semplice: è composta da una breve stringa che contiene istruzioni sul punto da cui partire a disegnare e sul colore. La prima istruzione è la BM (Blank Move) che posiziona la "matita" nel punto desiderato. La posizione di partenza è data dalle coordinate (23, 96): queste non sono racchiuse da parentesi come accade, invece, in altri comandi grafici. L'istruzione finale nella stringa è il colore di DRAW, C0 (nero). La linea 50 inizia a tracciare il profilo. La stringa che controlla il comando DRAW può sembrare caotica, ma è invece molto lineare: comprende una serie di direzioni e di distanze. Le lettere controllano la direzione della linea e i numeri danno la lunghezza in pixel. Se, dopo una direzione, non c'è un numero, lo spostamento è di un solo pixel. Si possono usare otto direzioni: D per basso, L per sinistra, R per destra, E per 45 gradi in alto a destra, F per 135 gradi in basso a destra, G per 225 gradi in basso a sinistra, H per 315 gradi in alto a sinistra.

A partire dalla linea 50, le stringhe spostano la "matita" a destra 28 pixel, a 45 gradi 2 pixel, in alto 3 pixel e così via. Traducendo la stringa in movimenti di una matita su un foglio, si comprende quale sarà il profilo della nave.

Le linee da 60 a 80 contengono stringhe simili, che completano il disegno. Si potrebbe usare soltanto una lunga stringa invece di tutte queste, ma ciò rende laborioso maneggiare e correggere i dati. Infine, il contorno viene colorato dalla linea 90, facendo apparire la nera sagoma della nave.

IL DISEGNO DI LETTERE

Un fastidioso limite, nella grafica del Dragon e del Tandy, è l'impossibilità di visualizzare contemporaneamente testi e grafica. Non si può, ad esempio, visualizzare il punteggio di un gioco, mentre si usa la grafica ad alta risoluzione. Per questo motivo il gioco a pagina 98 è interamente concepito su 'videate' di testo.

C'è però una soluzione al problema: i caratteri possono venir definiti e disegnati da DRAW. Poiché questo comando opera su stringhe, queste possono essere definite inizialmente e impiegate in un secondo DRAW.

```
1Ø PMODE 3,1
2Ø PLCS
3Ø SCREEN1,Ø
4Ø HE$="D4BR3U2NL3U2BR5L3D2NR2D2
   R3BR5L3U4BR5D4R3BR2U4R3D4L3"
5Ø DRAW"BM11Ø,5Ø;C3S8"+HE$
6Ø GOTO 6Ø
```

La sequenza contenuta in HE$ definisce le lettere per visualizzare "HELLO".

Come fatto in precedenza, si traduca la stringa in movimenti di una matita su un foglio, ricordando che B sta per "vuoto" e una direttiva preceduta da B non produce nessuna linea.

Volendo visualizzare più parole sullo schermo grafico (ad esempio: BUONA FORTUNA, BEL COLPO ecc.), si possono definire più stringhe (BF$ e BC$, per esempio). Si provi a formulare le corrette direttive per queste parole su carta quadrattata: le corrispondenti stringhe potranno essere usate nel programma.

"HELLO" viene disegnato alla linea 5Ø: la posizione di partenza è a 11Ø, 5Ø e il colore è 3 (blu), quindi comparirà una parola blu, in dimensione 8 (scala doppia).

S può essere seguito da un numero da 1 a 62, 1 è un quarto della grandezza, 4 è grandezza normale (quello selezionato in assenza di istruzioni esplicite) e 8 grandezza doppia. Come si vede nella linea 5Ø, le stringhe di DRAW si possono concatenare come normali stringhe. Volendo definire altre stringhe, come BF$ o BC$, queste andranno organizzate come alla linea 5Ø. Si raggiunga il punto di partenza con BM, poi si selezioni il colore con C e la dimensione con S.

MESSAGGI PIU' LUNGHI

Se il numero di messaggi da visualizzare è elevato, scrivere altrettante definizioni è un lavoro tedioso.

Conviene allora definire tutti i caratteri alfanumerici occorrenti e usarli con un programma del tipo:

```
1Ø PMODE 3,1
2Ø DIM LE$(26)
3Ø PCLS
4Ø FOR K=Ø TO 26:READ LE$(K):NEXT
5Ø FOR K=Ø TO 9:READ NU$(K):NEXT
6Ø DATA BR2,ND4R3D2NL3ND2BE2,ND4R3
   DGNL2FDNL3BU4BR2,NR3D4R3BU4BR2,
   ND4R2FD2GL2BE4BR,NR3D2NR2D2R3
   BU4BR2
7Ø DATA NR3D2NR2D2BE4BR,NR3D4R3U2
   LBE2BR,D4BR3U2NL3U2BR2,ND4BR2,BD4
   REU3L2R3BR2,D2ND2NF2E2BR2
8Ø DATA D4R3BU4BR2,ND4FREND4BR2,ND4
   F3DU4BR2,NR3D4R3U4BR2,ND4R3D2NL3
   BE2,NR3D4R3NHU4BR2
9Ø DATA ND4R3D2L2F2BU4BR2,BD4R3U2L3
   U2R3BR2,RND4RBR2,D4R2U4BR2,D3FEU3
   BR2,D4EFU4BR2
1ØØ DATA DF2DBL2UE2UBR2,DFND2EUBR2,
   R3G3DR3BU4BR2
11Ø DATA NR2D4R2U4BR2,BDEND4BR2,R2
   D2L2D2R2BU4BR2,NR2BD2NR2BD2R2U4
   BR2,D2R2D2U4BR2,NR2D2R2D2L2BE4,D4
   R2U2L2BE2BR2,R2ND4BR2,NR2D4R2U2
   NL2U2BR2,NR2D2R2D2U4BR2
12Ø SCREEN1,Ø
13Ø A$="VERIFICA Ø123456789"
14Ø DRAW"BM6Ø,5Ø;C3S8"
15Ø GOSUB 9ØØØ
16Ø GOTO 16Ø
9ØØØ FOR K=1 TO LEN(A$)
9Ø1Ø B$=MID$(A$,K,1)
9Ø2Ø IF B$>="Ø" AND B$<="9" THEN
   DRAW NU$(VAL(B$)):GOTO 9Ø5Ø
9Ø3Ø IF B$="□" THEN N=Ø ELSE N
   =ASC(B$)-64
9Ø4Ø DRAW LE$(N)
9Ø5Ø NEXT
9Ø6Ø RETURN
```

Le frasi DATA (linee da 6Ø a 11Ø) contengono la definizione di un set di caratteri composto dalle lettere maiuscole (da A a Z), dai numeri (da Ø a 9) e, importante da un carattere spazio. Il programma (linee 4Ø e 5Ø) legge le DATA, trasferendole nelle matrici LE$ e NU$.

Per usare il set di caratteri è necessario definire il messaggio, la posizione di partenza e altre informazioni, quali la dimensione e il colore. La variabile A$, nella linea 13Ø, contiene un messaggio di prova, che possiamo sostituire attraverso un'altra scritta.

La linea 14Ø assegna la posizione di partenza, il colore e la dimensione. Definito il messaggio e assegnato il punto di partenza, si può passare alla visualizzazione, eseguendo il programma. La subroutine di stampa, linea 9ØØØ, esamina ciascun carattere in A$, ne cerca la definizione nelle matrici e visualizza il carattere sullo schermo (vedere Giochi al Computer, pagine 144-147).

Si può cambiare messaggio a piacimento, modificando A$ (linea 13Ø) e, se necessario, la posizione (linea 14Ø). La parte di programma che visualizza le lettere è racchiusa in una subroutine, utilizzabile più volte nel programma, purché il messaggio sia sempre contenuto in A$.

Ciò è molto utile per visualizzare una serie di messaggi in vari fasi di un gioco. Il programma per la visualizzazione dei caratteri può essere inserito così com'è in tantissimi giochi: basta iniziare il gioco dalla linea 12Ø, o comunque dopo le DATA (eccezion fatta per le istruzioni DIM, CLEAR e PCLEAR, che devono trovarsi all'inizio). Quando occorre presentare un messaggio, basta prepararne il testo in una stringa A$ nel punto opportuno del programma, utilizzando poi DRAW nel modo visto alla linea 14Ø. Naturalmente, si rammenti di inserire la chiamata alla subroutine con GOSUB 9ØØØ.

E FINALMENTE

Avendo imparato l'uso dei comandi grafici, vediamo adesso come disporre le immagini diritte, capovolte o come ci pare. Basta aggiungere, nella stringa di DRAW, una A, seguita da un valore da Ø a 3.

AØ dispone il disegno a Ø gradi, cioè "diritto"; A1 sul fianco destro, a 9Ø gradi; A2 capovolto, a 18Ø gradi e A3 sul fianco sinistro a 27Ø gradi. Questo genere di controllo sull'immagine è molto utile per ottenere diversi effetti, partendo da una sola istruzione, anziché ripetere le definizioni come se si trattasse di un'immagine totalmente nuova. Per apprezzare il risultato, si lanci questo programma:

```
1Ø PMODE 3,1
2Ø PCLS
3Ø SCREEN 1,Ø
4Ø S$="NR16E8F4U4R2D6F2D12L6U6L4D6
   L6U12"
5Ø FOR K=1 TO 2Ø
6Ø D=RND (2ØØ)+27:E=RND (14Ø)+27:C
   =RND (3)+1:A=RND(4)-1
7Ø DRAW "BM"+STR$(D)+","+STR$(E)+
   "C"+STR$(C)+"A"+STR$(A)+"XS$;"
8Ø NEXT K
9Ø GOTO 9Ø
```

Si ottengono 2Ø case, ognuna identica alle altre, salvo il colore e l'orientamento.

La forma delle case è definita dalla stringa nella linea 4Ø. La linea 6Ø sceglie 4 numeri a caso: le coordinate D e E, il colore C e l'angolo di orientamento A.

La linea 7Ø disegna la casa combinando assieme gli elementi della stringa e i numeri casuali. STR$ converte le variabili numeriche in stringhe, in pratica circondando con le virgolette il valore delle variabili numeriche: se D=2, allora STR$ (D) ="2".

La linea 7Ø sposta il cursore in un punto casuale di partenza, seleziona il colore scelto a caso, e disegna l'immagine ad un'angolazione anch'essa casuale. La X, prima di S$, serve per concatenare, nella DRAW, la stringa S$. Prima avevamo usato, col medesimo effetto, +HE$.

Questo metodo di concatenamento è molto "economico", perché consente di riunire più stringhe in una sola, senza peraltro impegnare ulteriore memoria con una nuova variabile stringa.

INDICE CUMULATIVO DEI FASCICOLI

NEL PROSS

☐ *Scopriamo come è fatta la MEMORIA del nostro computer e cosa accade quando depositiamo in essa i programmi in codice macchina.*

☐ *Nella sezione Giochi al Computer, è spiegato come aggiungere più LIVELLI DI DIFFICOLTÀ ai nostri programmi di gioco, oltre a un programma per labirinti.*

☐ *Le sequenze di caratteri nelle variabili stringa si possono elaborare efficacemente mediante le FUNZIONI STRINGA.*

☐ *Un esempio pratico di PROGRAMMAZIONE STRUTTURATA: lo sviluppo completo di un programma di ordinamento "Bubble sort".*

☐ *I JOYSTICK costituiscono un economico sistema alternativo alla tastiera per comunicare con il computer, ma non servono soltanto nei giochi.*

CHIEDETE INPUT AI